Skaalautuvan tietokannan suunnittelu ja toteutus hankintailmoitusportaaliin

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Hämeen ammattikorkeakoulun opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

Visamäki, syksy 2013

Kari Lehtomaa

Kari Lehtomaa

HAMK
HÄMEEN AMMATTIKORKEAKOULU

TIIVISTELMÄ

Visamäki
Tietojenkäsittely
Suuntautumisvaihtoehto

Tekijä Kari Lehtomaa **Vuosi** 2013

Työn nimi Skaalautuvan tietokannan suunnittelu ja toteutus hankintailmoitusportaaliin

TIIVISTELMÄ

Työn toimeksiantaja Seravo Oy on pieni muutaman työntekijän tamperelainen ohjelmistoalan yritys, joka suunnittelee, toteuttaa ja ylläpitää avoimen lähdekoodin ohjelmistoja ja palveluja. Seravo Oy oli toteuttamassa avointa hankintailmoitusportaalia, joka käyttäisi nykyaikaisia tietokanta- ja verkkosovellustekniikoita.

Työnä oli skaalautuvan tietokannan suunnittelu ja toteutus hankintailmoitusportaaliin. Työn tavoitteena oli tutkia NoSQL-tietokantojen keskeisiä periaatteita ja valita sopiva tietokanta kahdesta vaihtoehdosta Node.js-ympäristölle tehdylle hankintailmoitusportaalille. Olennaista työssä oli, että täytyi myös perehtyä verkkosovellusten kehitykseen Node.js-ympäristössä.

Käytännön osuus työstä oli luonteeltaan sovelluskehitys-toimeksianto, jossa hankintailmoitusportaalille tuli kehittää ja toteuttaa kansallisten hankintailmoitusten noutosovellus sekä haku- ja ilmoitustoiminnot olemassa olevaan uuteen portaaliin. Teoria-aineistona oli pääasiassa Internet-lähteitä, koska käytettävät teknologiat ovat nopeasti kehittyviä. Teorian oikeellisuus varmistettiin käyttämällä samaan asiaan useita eri aineistoja. Sovelluskehitys-vaiheessa oli käytettävissä myös virallisen portaalin rajapintakuvauksia ja muuta teknistä materiaalia. Kehityksen aikana sovelluksen koodi on suljettua koodia, joten siitä esitetään vain pieniä otteita.

Vaaditut toiminnot saatiin toteutettua ja samalla saatiin kokemusta ja tietämystä NoSQL-tietokannoista ja niiden käyttämisestä verkkosovelluksessa. Node.js todettiin toimivaksi ympäristöksi, mutta samalla löydettiin myös ongelmakohtia. Tuotantokäyttöä varten toiminnot vaativat vielä hiomista.

Avainsanat hankintailmoitus, hajautus, tietokanta, NoSQL, Node.js, ohjelmointi.

Sivut 36 s. + liitteet 10 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES

HAMK Visamäki
Degree Programme in Business Information Technology
System engineering

Author Kari Lehtomaa **Year** 2013

Subject of Bachelor's thesis **Implementing scalable database for procurement notice portal.**

ABSTRACT

This thesis was commissioned by a small company Seravo Oy in Tampere. Seravo operates in software industry using mainly open source software. Seravo design, implement and do maintenance using open source. The company was implementing a web portal for national procurement notices, which will use the newest web and database technologies.

The main objective of this thesis was to design and implement scalable distributed database for a procurement notice web portal. The main objective was to study NoSQL databases and their essential features, and select appropriate database from two candidates. In addition, the aim was to develop and implement a notice collection function, search function and notification function to Node.js based web portal.

Internet sources were used for collecting theoretical information, but also some books and interface documents from the author of the official procurement noticeportal were used. The correctness of the sources was ensured by using different sources for the same thing. During the development the new web portal is a closed source code and therefore only small bits of code are presented.

The required functions were successfully implemented in the thesis. The functions work correctly, but they are not ready for production use. During the programming plenty of challenges and problems were encountered. As a result of the study a lot of new knowledge about NoSQL and Node,js.was obtained.

Keywords Procurement notice, distributed systems, database, NoSQL, Node.js, software development.

Pages 36 p. + appendices 10 p.

SISÄLLYS

1 JOHDANTO....1

2 HANKINTAILMOITUSPORTAALI....2
2.1 Julkinen hankinta....2
2.2 Ongelmat HILMAssa....3
2.3 Uusi portaali....3

3 NOSQL-TIETOKANNAT....4
3.1 Skaalautuminen....7
3.2 Tietokantatyypit....8
3.3 NoSQL-tietokantojen ominaispiirteitä....11
3.4 Tiedonkäsittely Map/Reducella....13
3.5 Riak....14
3.5.1 HTTP-rajapinta....15
3.5.2 Hakutoiminnot....16
3.6 Couchbase....18

4 OHJELMOINTIYMPÄRISTÖ NODE.JS....20
4.1 Tiedon kuvausmuoto JSON....22
4.2 Lisäkirjastot....23
4.3 Verkko-ohjelmointi....23

5 TOIMINTOJEN SUUNNITTELU JA TOTEUTUS....25
5.1 Git-versionhallinta....25
5.2 Tietokanta....26
5.3 Noutotoiminto....27
5.4 Hakutoiminto....30
5.5 Ilmoitustoiminto....31
5.6 Hajautuksen testaus....32
5.7 Yhteenveto toteutuksesta....33

6 YHTEENVETO....34

7 LÄHTEET....35

Liite 1 Node.js Express-malli
Liite 2 Tiedontallennus Couchbaseen Node.js:llä.
Liite 3 Tiedontallennus Riakiin Node.js:llä.
Liite 4 Hankintailmoitus XML-muodossa.
Liite 5 Määritys XML-mapping.json-tiedostossa.
Liite 6 Hakutoiminnon Map/Reduce.

TERMIT JA LYHENTEET

ACID	Atomicity, Consistency, Isolation, Durability. Relaatiotietokantoihin liittyvä periaate, jolla turvataan tietojen eheys.
AJAX	Asynchronous JavaScript And XML. Javaskripti ja XML-pohjainen verkkosovelluskehitys-tekniikka, joka mahdollistaa vuorovaikutteiset verkkosovellukset.
Apache	Internetin suosituin HTTP-palvelin, joka perustuu avoimeen lähdekoodiin.
API	Application Programming Interface. Ohjelmointirajapinta, jonka välityksellä eri ohjelmat voivat keskustella keskenään.
BASE	Basically Available, Soft state, Eventual consistency. NoSQL-tietokantojen eheyttä kuvaava käsite.
Big data	Käsite suurille tietomäärille, jota ei voida käsitellä perinteisellä tavalla.
CAP-teoreema	Consistency Availability Partition tolerance. Teoreettinen hajautetun tietokannan vaatimuskuvaus.
CPV	Common Procurement Vocabulary. EU:n hyväksymä hankintasanaston luokittelujärjestelmä, jota käytetään hankintailmoituksissa yksilöimään hankinnan tuotetta tai palvelua.
Curl	Komentorivipohjainen HTTP-asiakasohjelma
DNS	Domain Name System. Internetin nimipalvelujärjestelmä
Erlang	Funktionaalinen ohjelmointikieli.
Git	Linux Torvaldsin kehittämä avoimen lähdekoodin hajautettu versionhallintaohjelmisto
HILMA	Virallinen tietokanta kotimaan hankintailmoituksille
HTML5	Moderni ja uusin versio verkkosivuston luontiin tarkoitetusta kuvauskielestä
HTTP	Hypertext Transfer Protocol. Protokolla WWW-tiedonsiirtoon

JDBC	Java Database Connectivity. Java-ohjelmointikielen tietokantarajapinta
JSON	JavaScript Object Notation. Javaskriptissä käytettävä tietorakennemuoto.
JQuery	Javaskriptikirjasto, joka helpottaa Javaskripti-pohjaisten toimintojen toteutusta.
Jquery UI	Javaskriptipohjainen internet-sivujen käyttöliittymäkirjasto .
Lucene	Avoimen lähdekoodin tiedonhakukirjasto
Node.js	Javaskripti-pohjainen ohjelmistoalusta
NPM	Node Package Manager. Node.js lisäkirjastojen hallintatyökalu.
NUTS	Nomenclature of Territorial Units for Statistics. Hierarkinen aluekoodi, jonka avulla voidaan hakea tietyn alueen ilmoituksia HILMAssa ja TEDissä.
ODBC	Open Database Connectivity. Standardoitu avoin tietokantarajapinta.
REST	Representational State Transfer. Ohjelmointirajapintojen toteuttamiseen tarkoitettu HTTP-protokollaan perustuva arkkitehtuurimalli.
SQL	Relaatiotietokantojen kyselykieli
TED	Tenders Electronic Daily. Tietokanta EU-alueen hankintailmoituksille.
Twitter bootstrap	Ulkoasun HTML5-sovelluskehys
Virtualbox	Oraclen valmistama avoimen lähdekoodin virtualisointialusta
XML	Extensible Markup Language. Rakenteellinen kuvauskieli.
XPath	Kieli XML-dokumentin osien osoittamiseen
XSD	Standardi tekniikka, jolla kuvataan XML-tiedostojen rakenne

1 JOHDANTO

Julkisia hankintoja varten on olemassa virallinen hankintailmoitusportaali, johon erilaiset hankintayksiköt ilmoittavat omista hankinnoistaan. Kuitenkin tämän työn toimeksiantaja Seravo Oy on katsonut, että uusi palvelu olisi tarpeen, koska virallinen hankintailmoitus- eli HILMA-palvelu ei tarjoa tarpeeksi toimintoja tai riittävän helppoa ja tehokasta ratkaisua.

Työn toimeksiantaja Seravo Oy on pieni tamperelainen ohjelmistoalan yritys, joka suunnittelee, toteuttaa ja ylläpitää avoimen lähdekoodin ohjelmistoja ja palveluja. Seravo Oy:n teknistä erityisosaamista ovat avoimen lähdekoodin hyödyntäminen tuotteissa ja palveluissa, Linux-tukipalvelut, virtualisointi ja HTML5-sovellukset.

Seravo oli jo aiemmin aloittanut uuden palvelun toteuttamisen, mutta siitä puuttui vielä tehokas skaalautuva tietokanta, hakutoiminnot ja uusien hankintailmoitusten ilmoitustoiminto. Lisäksi olemassaoleva hankintailmoitusten noutotoiminto vaatii uudelleen toteuttamisen. Työ on luonteeltaan kehitystehtävä ja sen keskeinen idea on tutkia ja valita sovellukselle tietokantaratkaisu sekä suunnitella ja toteuttaa tietojen nouto-, haku- sekä ilmoitustoiminnot.

Toimeksiantaja oli jo etukäteen päättänyt, että sovellus tulee käyttämään NoSQL-tietokantaa. Tietokannan valintaa varten perehdytään yleisesti NoSQL-tietokantoihin ja tutkitaan tarkemmin kahden ehdokkaan Riakin ja Couchbasen sopivuutta sovelluksen tarpeisiin. Työssä ei perehdytä tietokantojen tietoturvaan tai varmistukseen. Uusi palvelu on toteutettu Node.js-ympäristöllä. Työssä esitellään myös Node.js:n toimintaperiaatteita sekä kuvataan lyhyesti sen käyttöä verkkosovellusten kehitykseen.

Työ on ajankohtainen, koska siinä tutkitaan uusien tekniikoiden NoSQL:n ja Node.js:n käyttöä. Varsinkaan Suomessa näiden tekniikoiden käyttäminen ja yleinen tietämys ei ole kovin laajaa, joten työ tuottaa uutta suomenkielistä tietoa näistä aihepiireistä. Sovelluskehitystyö toteutetaan käyttämällä pelkästään avoimen lähdekoodin ohjelmistoja. Työhön liittyy useita teknisiä käsitteitä, kuten esimerkiksi XML, mutta niihin ei perehdytty tässä työssä. Työn teknisyyden vuoksi työ on suunnattu pääasiassa tietotekniikkaa tunteville.

Työn tärkeimpinä kysymyksinä on selvittää NoSQL-tietokantojen keskeisimmät käsitteet ja toimintaperiaatteet, Riakin tai Couchbasen sopivuus palvelua varten ja vaadittujen toimintojen toteutus Node.js-ympäristöllä.

2 HANKINTAILMOITUSPORTAALI

Hankintailmoitusportaali eli HILMA on työ- ja elinkeinoministeriön ylläpitämä maksuton ja verkkopohjainen virallinen ilmoituskanava, jossa hankintayksiköt ilmoittavat julkisista hankinnoistaan. HILMAssa ilmoitetaan sekä kansallisen että EU:n kynnysarvon ylittävät hankinnat. Uusia hankintoja voidaan ilmoittaa rekisteröitymällä ensin käyttäjäksi. Voimassaolevia ilmoituksia voi selata tai hakea luettavaksi ilman rekisteröitymistä . HILMAssa on myös saatavilla eri vuosien tilastoja sekä tietoa julkisista hankinnoista yleisesti. Hankintailmoitusten seurantaan on myös muita verkkopalveluja, kuten esimerkiksi www.credita.fi. Ne ovat yleensä monipuolisempia, mutta niiden käyttö voi olla maksullista. (HILMA 2013.)

2.1 Julkinen hankinta

Julkiset hankinnat ovat erilaisten julkisten hankintayksiköiden, kuten esimerkiksi valtionyritykset, kunnat sekä seurakunnat, tekemiä hankintoja oman organisaationsa ulkopuolelta. Julkiset hankinnat tulee tehdä voimassaolevan lainsäädännön mukaisesti. Tästä syystä hankinnat tulee kilpailuttaa avoimesti ja kilpailuun osallistuvia yrityksiä ei saa syrjiä. (Julkinen hankinta 2013.)

Hankinnan ennakoitu arvo voi ylittää EU-kynnysarvon, jolloin hankinnat pitää toimittaa HILMAn lisäksi automaattisesti julkaistavaksi EU:n TED-tietokannassa, joka sisältää kaikki EU-alueen EU-kynnysarvon ylittävät hankintailmoitukset. Kansallisen kynnysarvon ylittävät hankinnat pitää ilmoittaa HILMAssa, mutta sen alittaville hankinnoille hankintayksiköt voivat toimia oman harkintansa mukaan. (Julkinen hankinta 2013.)

Laissa sanotaan hankintaa koskevista ilmoituksista, että hankintayksikön on ilmoitettava julkisesti ennakkoilmoitus, hankintailmoitus ja suunnittelukilpailun ilmoitus, käyttöoikeusurakkaa koskeva ilmoitus sekä jälki-ilmoitus. Ilmoitukset tulee toimittaa julkaistavaksi työ- ja elinkeinoministeriön määräämälle taholle. (Hankintalaki 6:35 §.)

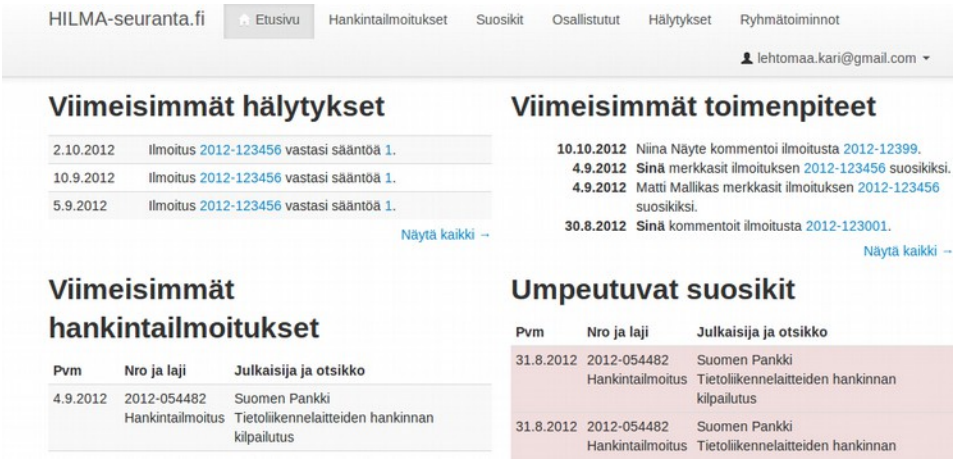
Valtioneuvoston asetuksessa sanotaan ilmoitusvelvollisuudesta, että ilmoitusvelvollisuus on kansallisen kynnysarvon ylittävissä hankinnoissa, toissijaisissa palveluhankinnoissa ja eräissä muissa hankinnoissa. Hankintalain 68 §:n 1 momentissa edellytetyt kansalliset ilmoitukset on toimitettava julkaistaviksi Internet-osoitteessa www.hankintailmoitukset.fi. (Valtioneuvosto 614/2007.)

2.2 Ongelmat HILMAssa

Toimeksiantajan mukaan virallinen hankintailmoitusportaali on yksinkertainen ja toiminnoiltaan puutteellinen. Hankintailmoituksia työkseen seuraavalle tarjotaan vain pelkkä haku tai ilmoitusten selailu uutuuslistalta sitä mukaa kuin niitä ilmestyy. HILMAssa ei voi myöskään selailla ilmoituksia, joiden hankinta-aika on päättynyt . Käyttäjä ei voi merkitä ja pyytää itselleen kiinnostavia ilmoituksia esimerkiksi CPV- tai NUTS-koodin mukaan. Käyttäjä ei voi saada automaattisesti tietoa uusista ilmoituksista eikä palvelussa voi myöskään hyödyntää ryhmätyöominaisuuksia.

2.3 Uusi portaali

Uuden portaalin kehitys oli aloitettu käyttämällä nykyaikaisia HTML5-tekniikoita. Sovelluksen palvelimella suoritettava osa on toteutettu Node.js -ympäristöllä, selaimessa oleva osa käyttää Javaskriptiä ja ulkoasun sovelluskehyksenä on Twitter bootstrap. Twitter bootstrap on HTML5-sovelluskehys, joka avulla voidaan luoda helpolla tavalla hyvännäköinen selkeä käyttöliittymä, joka toimii samalla tavalla selaimesta riippumatta ja myös mobiililaitteissa. Kuvassa 1. on esitetty sovelluksen käyttöliittymän ulkoasua käyttäjän kirjautumisen jälkeen.



Kuva 1. Uuden portaalin käyttöliittymää

Verrattuna virallisen portaalin ominaisuuksiin uuteen sovellukseen toteutetaan:

1. Skaalautuva tietokanta
2. Kehittyneet hakutoiminnot
3. Käyttäjäkohtaiset ilmoitukset käyttäjää kiinnostavista hankinnoista käyttöliittymässä ja sähköpostitse.
4. Kommentointi ja muut ryhmätyöominaisuudet

Toimintojen lisäksi palvelua varten toteutetaan erillisenä noutotoiminto, joka noutaa hankintailmoituksia tietokantaan ohjelmallisesti HTTP-protokollalla XML-muodossa. Rajapinnan kautta saa haettua vain voimassa olevia ilmoituksia eli niitä, jotka näkyvät virallisen hankintailmoitusportaalin kautta. HILMAssa arkistoitujen vanhojen ilmoitusten haku rajapinnan kautta ei onnistu. Taulukossa 1. on listattu kaikki rajapinnan kautta saatavilla olevat hankintailmoitustyypit. (Rajapintamäärittely 2011.)

Taulukko 1. Hankintailmoitustyypit

Englanninkielinen nimitys	**Suomenkielinen nimitys**
DOMESTIC_CONTRACT	Kansallinen ilmoitus
PRIOR_INFORMATION	Ennakkoilmoitus
CONTRACT	Hankintailmoitus
CONTRACT_AWARD	Ilmoitus hankintasopimuksesta
CONTRACT_UTILITIES	Hankintasopimus, erityisalat
CONTRACT_AWARD_UTILITIES	Ilmoitus hankintasopimuksesta, erityisalat
SIMPLIFIED_CONTRACT	Dynaamista hankintajärjestelmää koskeva lyhennetty hankintailmoitus
VOLUNTARY_EX_ANTE_TRANSPARENCY_NOTICE	Suorahankintaa koskeva ilmoitus

Tärkeä ominaisuus uudessa portaalissa on se, että se toteutetaan käyttämällä skaalautuvia teknologioita. Virallisen portaalin tietokanta-ratkaisusta ei ole tietoa, mutta oletettavasti se käyttää perinteistä relaatiotietokantaa. Skaalautuvien teknologioiden myötä portaalia voidaan laajentaa kattamaan myös koko EU:n alueen hankintailmoitukset, jolloin palvelua voidaan käyttää koko EU:n alueella.

3 NOSQL-TIETOKANNAT

Relaatio- eli SQL-tietokannat ovat olleet käytössä vuosikymmeniä ja ne ovat kehittyneet palvelemaan tehtävässään. Kuitenkin tiedonmäärän nopean kasvun, pilvipohjaisten palveluiden ja erityisesti maailmanlaajuisten verkkosovellusten vaatimusten takia relaatiotietokannat eivät sovi kaikkiin tarkoituksiin kovin hyvin. Tiedonmäärän valtavasta kasvusta saa jonkinlaisen käsityksen Spencer Neilin blogikirjoituksesta, jonka mukaan esimerkiksi joka minuutti Youtubeen ladataan 48-tuntia videota ja Facebookiin luodaan sisältöä lähes 700 000 kappaletta (Spencer 2012). Käytettävä tietokanta voi olla yritykselle taloudellinen tekijä, koska suurien tietomäärien käsittelyssä relaatiotietokannat voivat muuttua hitaiksi ja sen myötä sovel-

luksen käyttäjät siirtyvät käyttämään kilpailijan palvelua. Kustannustehokas ratkaisu on ollut kehittää tietynlaisia skaalautuvia tietokantoja nimenomaan suurien tietomäärien tallennukseen. (Why NoSQL n.d.)

NoSQL:n katsotaan alkaneen vuodesta 2009, joten se on käsitteenä varsin uusi tietokantamalli, mutta se on yleistynyt voimakkaasti. Se voidaan määritellä termeillä ei-relaatio, hajautettu ja horisontaalisesti skaalautuva. Lisäksi tietokannat ovat yleensä avoimen lähdekoodin tuotteita. (NoSQL n.d.)

NoSQL-tietokannat ovat suurien tietomäärien tietokantoja. Kun perehdytään isojen tietomäärien käsittelyyn, törmätään aina termeihin Bigdata-teknologia ja NoSQL-tietokannat. Määritelmällä Bigdata ei ole tiettyä tavumäärää, vaan esimerkiksi 100 Gigatavua voi olla jo Bigdataa, jos tiedon rakenne tai sen analysointi on monimutkaista. Suuret tietomäärät voivat vaatia jopa tuhansien koneiden ympäristöjä, joissa yksittäistä konetta kutsutaan solmuksi, engl. Node. Suurten tietomäärien ominaisuuksista johtuen tietokannoilla tulee olla tiettyjä ominaisuuksia, kuten Gupta esittää (Gupta V. 2010.)

1. Korkea skaalautuvuus, jolloin järjestelmään voidaan lisätä suuri määrä solmukoneita.
2. Korkea käytettävyys, jossa tietokanta toimii, vaikka jotkin solmukoneet eivät olisi toiminnassa.
3. Korkea suorituskyky
4. Joustavuus sekä palvelimien lisäyksessä että tiedon mallinnuksessa.

NoSQL-tietokannoilla ei ole etukäteen määriteltyä kiinteää tiedon rakennetta, kuten relaatiotietokantojen tauluilla on. Rakennetta voidaan muuttaa helposti sitä käyttävän sovelluksen muuttuessa. Ominaisuuksiin kuuluu myös nopea tiedon nouto avaimella ja korkea kirjoitusnopeus suurilla tietomäärillä. Relaatiotietokantojen taulujen kiinteä rakenne voi olla ongelma, jos sovellukseen halutaan tehdä uusia ominaisuuksia tai tiedon rakenne ei ole yhtenäinen. Relaatiotietokannan taulun rakenteen muuttaminen jälkikäteen voi olla erittäin työlästä. Hierarkkisen tiedon esittäminen on relaatiotietokannalla vaikeaa varsinkin monimutkaisemman tietomallin tapauksessa. Joustavalla rakenteella saavutetaan etuna nopeampi ohjelmistokehitys etenkin ketterissä menetelmissä, joissa koko sovellusta ei määritellä täysin valmiiksi etukäteen vaan määritelmä muuttuu sovelluksen toteutuksen yhteydessä. (Gupta 2010.)

NoSQL-tietokannat eivät täysin korvaa relaatiotietokantoja vaan ovat pikemminkin täydentävä tekijä. Esimerkiksi järjestelmissä, joissa tarvitaan paljon laskentaa, tiedolla on selkeä tiivis rakenne tai tietojen eheys on tär-

keää relaatiotietokanta on useimmiten oikea valinta. Raportoinnin ja analysoinnin kannalta relaatiotietokannoilla on se etu, että niihin on saatavilla ODBC- ja JDBC-rajapinnat, kun taas NoSQL-tietokannoille ei ole näitä rajapintoja saatavilla. NoSQL-tietokannoilla on vastaavasti yleensä joko HTTP-protokollaan perustuva REST-rajapinta tai muu vastaava. REST-rajapinnan etuna verrattuna esimerkiksi JDBC:hen on sen käyttö ilman lisäkirjastoja, koska REST on HTTP-protokollaa. Lisäksi NoSQL-tietokannoille on tyypillistä monipuolinen tuki eri ohjelmointikielille, kuten voidaan havaita tietokantoja esittävällä sivustolla (NoSQL n.d.). Relaatiotietokannoilla on yhteinen SQL-kyselykieli, kun taas NoSQL-tietokannat eivät käytä SQL-kyselykieltä. NoSQL-tietokantatuotteilla on yleensä jokaisella omat ratkaisunsa kyselyjen tekemiseen. Yleisesti kaikkia yhdistävä ratkaisu tiedon hakuun ja suodattamiseen on Map/Reduce, johon perehdytään tarkemmin kappaleessa 3.4.

NoSQL-tietokannat eivät siis sisällä transaktioita, sarakekohtaista indeksointia tai taulujen välisiä liitoksia. Tietokannat ovat siis vain tiedon tallentamista ja hajautusta varten, joten loogiset operaatiot jätetään pääosin sovelluksen vastuulle. Tällöin tietokanta on rakenteeltaan yksinkertaisempi. Tietokanta toimii nopeammin, se tarvitsee vähemmän konfigurointia ja ylläpidollisia toimenpiteitä. Taulukossa 2 on esitetty SQL:n ja NoSQL:n pääominaisuudet yhteenvetona.

Taulukko 2. SQL ja NoSQL ominaisuusyhteenveto.

Ominaisuus	Relaatiotietokanta	NoSQL
Transaktiot	Kyllä	Ei
Kyselykieli	SQL	Valmistajakohtainen, Map/Reduce
Skaalautuvuus	Horisontaalisesti vaikeaa ja kallista, vertikaalinen mahdollista.	Horisontaalisesti yksinkertaista.
Rajapinnat	ODBC, JDBC	REST, valmistajakohtainen
Lisenssi	Yleisesti kaupallinen, mutta myös joitain avoimen lähdekoodin tuotteita.	Avoin lähdekoodi
Tallennetun tiedon rakenne	Etukäteen määritelty, hankalasti muutettava	Helposti muunnettava.

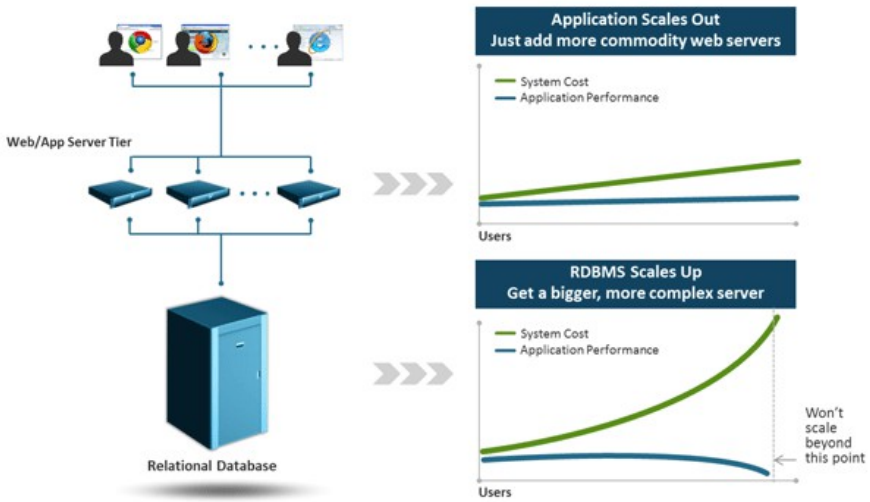
Suuria maailmanlaajuisia palveluja tarjoavat yritykset, kuten Google, Facebook ja Amazon käyttävät ja ovat kehittäneet omia Bigdata ja NoSQL-ratkaisuja. Google on kehittänyt Bigtablen ja Amazon Dynamon. Yleisin käyttötapa NoSQL-tietokannoille on Internet-verkkosovellusten tietokan-

toina.

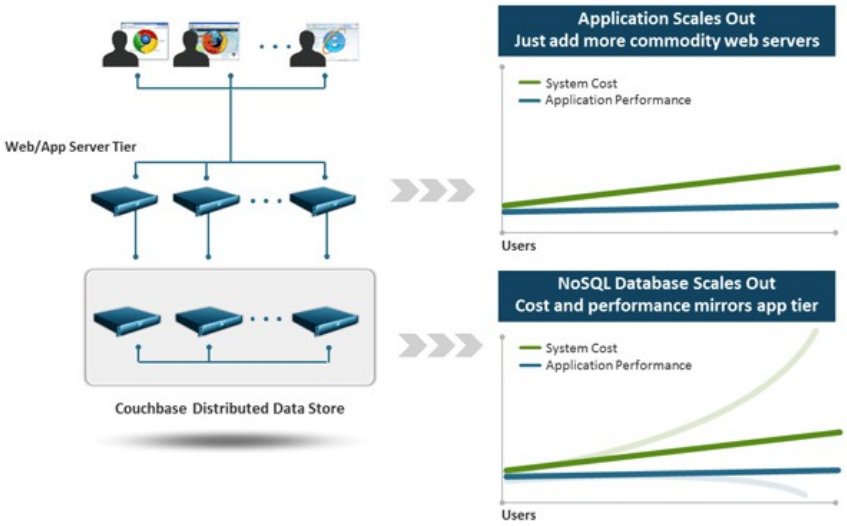
3.1 Skaalautuminen

Sovelluksen ja tietokannan muodostamaa järjestelmää voidaan joutua skaalaamaan, kun sen käyttäjämäärä lisääntyy tai järjestelmään lisätään uusia ominaisuuksia. Järjestelmä tai palvelu voi skaalautua monella eri tavalla. Tietokantojen yhteydessä skaalautumisella tarkoitetaan tietokannan suorituskyvyn tai tallennustilan nostamista. Perinteisesti relaatietokantapohjainen järjestelmä voi skaalautua ylöspäin eli vertikaalisti, jolloin palvelimen tehoa nostetaan lisäämällä muistia tai vaihtamalla laitteisto tehokkaampaan. Toinen vaihtoehto on skaalautua horisontaalisesti, mikä tarkoittaa, että palvelimien määrää kasvatetaan ja silti järjestelmä toimii yhtenä loogisena yksikkönä. (Adamo 2011.)

Horisontaalinen skaalautuminen voidaan toteuttaa replikoimalla, jolloin samasta tiedosta on useita kopioita eri koneilla. Tällöin puhutaan käytettävyyden kasvattamisesta. Se voi skaalautua myös osittamalla, jolloin tieto pilkotaan usealle tietokoneelle. Tällöin kasvatetaan tietokannan tehokkuutta ja tallennustilaa. Molempien tekniikoiden käyttö on relaatiotietokantojen yhteydessä yleensä monimutkaista ja kallista. Kuvassa 2 esitetään, kuinka relaatiotietokantojen skaalaus nostaa järjestelmän kustannuksia ja tietyssä pisteessä skaalautuminen ei ole enää järkevää tai se ei enää ole teknisesti mahdollista. Kuvassa 3 esitetään vastaava kaavio NoSQL-tietokannoille, joille skaalautumisen kustannukset kasvavat tasaisesti. Yleisesti NoSQL-tietokantojen skaalautuminen on valmiiksi järjestelmään rakennettuna. (Gupta 2010.)



Kuva 2. Sovellus skaalautuu, Relaatiotietokanta ei (Why NoSQL n.d.).



Kuva 3. Sovellus skaalautuu, NoSQL skaalautuu (Why NoSQL n.d.).

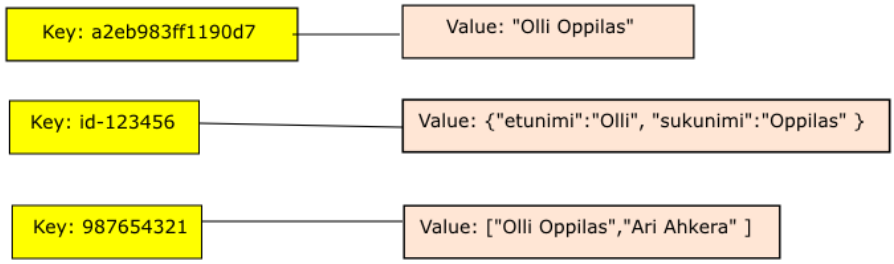
3.2 Tietokantatyypit

NoSQL-tietokantoja esittävällä sivustolla on listattu noin 150 erilaista tietokantaa. Ne voidaan jakaa moneen eriin tyyppiin tietomallin tai tiedontallennustavan mukaan. Tietokantojen tallennustapa voi olla tallennus fyysiselle levylle, pelkästään muistiin tai se voi olla yhdistelmä näistä molemmista. Muistinvaraiset tietokannat ovat tietenkin erittäin nopeita, mutta tieto katoaa, jos palvelin sammutetaan. Muistinvaraisia tietokantoja, kuten Redis, käytetään paljon web-sovellusten istuntomuisteina sekä välimuisteina muille tietokannoille nopeuttamaan kyselyitä. Tietokannan tyyppi ilmaisee sen, miten tallennettu tieto esitetään tietokannassa. Neljä yleisintä tyyppiä ja muutama niihin kuuluva tietokanta on esitetty alla olevassa taulukossa 2. (NoSQL n.d.)

Taulukko 3.NoSQL-tietokanta tyypit.

Tyyppi	Tietokanta	Tallennustapa
Avain-arvo, Key-value	Riak Redis Couchbase	Levy Muisti Levy ja muisti
Dokumentti, Document-store	Couchbase MongoDB	Levy ja muisti Levy
Sarake, Column family	Hadoop Hbase Cassandra Google Bigtable	Levy ja muisti Levy ja muisti Levy ja muisti
Verkkotietokanta, Graph database	Neo4J	Levy ja muisti

Avain-arvo tietokannoissa tieto esitetään aina nimensä mukaisesti avain- ja arvopareina engl. Key-Value tai KV. Avain on aina yksilöllinen tunniste, jonka tietokanta muodostaa itse tai se annetaan tallennuksen yhteydessä. Arvo voi olla yksittäinen merkkijono tai numero ja joissain tuotteissa se voi olla myös JSON-tietue tai taulukko, kuten kuvassa 4 esitetään. Kuvassa ylimpänä on yksittäinen arvo, sen alla JSON ja alimpana taulukkomuotoinen arvo. JSONista kerrotaan tarkemmin Node.js:n yhteydessä kappaleessa 4. Tätä tyyppiä edustaa mm. Riak ja Redis. Riakiin tutustutaan syvällisemmin kappaleessa 3.5.



Kuva 4. Avain-arvo tietosisältöjen mahdollisuuksia.

Dokumenttimuotoisissa tietokannoissa tieto esitetään JSON-muotoisina dokumentteina, kuten kuvassa 5 esitetään. Dokumentissa on aina jokin avainkenttä, joka voidaan itse määritellä tai järjestelmä muodostaa sen. Kuvassa 5 avainkenttänä on _id ja _rev-kenttää käytetään samanaikaisuuden hallintaan eli dokumenttejä voidaan versioida. Näitä kenttiä kutsutaan myös dokumentin metatiedoiksi. Dokumenttimalli sopii arvo-avainvarastoa paremmin monimutkaisten tietorakenteiden esittämiseen, koska dokumenttimallissa kenttiin voidaan viitata muodossa: dokumentti.etunimi. Avain-arvo-mallissa kenttä on arvon sisällä, jolloin siihen ei voida tällä tavoin viitata. Yleisimpiä dokumenttipohjaisia tietokantoja on Couchdb, Couchbase ja MongoDB. Myös jo ennen NoSQL-aikakautta olemassa ollut Lotus Notes kuuluu tähän ryhmään. Couchbase-tietokantaan perehdytään tarkemmin kappaleessa 3.6. (Anderson, Lehnardt, Slater n.d.)

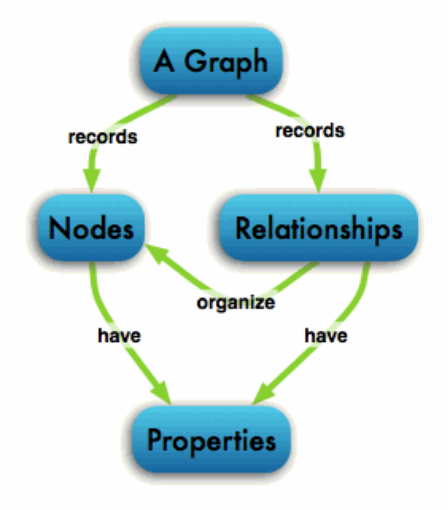
Kuva 5. Couchdb-dokumenttimalli.

Saraketietokanta on avain-arvotietokannasta kehittyneempi versio. Tiedot ovat riveillä, joilla on riviavain engl. Row-key, aikaleima ja riviin kuuluvia sarakeperheitä, engl. Column family. Sarakeperheisiin kuuluu itse sarakkeet. Aikaleima mahdollistaa tietojen versioinnin. Malli on esitetty havainnollisesti alla olevassa taulukossa 2. Selkeästi tätä mallia toteuttaa Hbase. (Hbase 2012.)

Taulukko 4. Saraketietokannan tietomalli.

Rivi avain	Sarakeperhe: nimi	Sarakeperhe: osoite	Aikaleima
K1234	Nimi:etunimi =Ari Nimi:sukunimi = Ahkera	Osoite:katu = Tietokatu 1	1234567899
K2345	Nimi:etunimi = Olli Nimi:sukunimi = Oppilas	Osoite:katu = Kantakatu 2	1234599999

Verkkotietokannat engl. Graph-databases poikkeavat tiedon esitykseltään huomattavasti edellä mainituista. Verkkotietokannoissa keskeistä on tietojen linkittyminen toisiinsa. Tietojen väliset suhteet ovat tärkeitä tiedon sisällön lisäksi. Kuvassa 6. esitetään havainnollinen malli, miten verkkotietokanta koostuu solmuista, niiden ominaisuuksista ja suhteista. (Neo4J n.d.)



Kuva 6. Neo4J tietomalli (Neo4J n.d.)

3.3 NoSQL-tietokantojen ominaispiirteitä

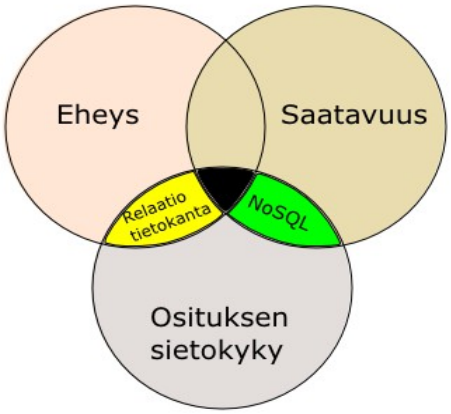
Relaatiotietokantojen tiedonkäsittelyyn erityisesti transaktioiden kohdalle kuuluu käsite ACID. ACID tarkoittaa sitä, että tietokantatapahtumat ovat jakamattomia, tietokanta säilyy eheänä toiminnon jälkeen, tapahtumat ovat eristettyjä toisistaan ja muutokset tietokantaan ovat pysyviä. Hajautuksen ja skaalautuvuuden takia NoSQL-tietokannat eivät voi toimia tämän periaatteen mukaisesti vaan niiden yhteydessä käytetään käsitettä BASE. Se tarkoittaa, että tiedot ovat periaatteessa käytettävissä, ne eivät ole aina oikeellisia ja ovat lopulta oikeellisia. NoSQL-tietokanta voi siis olla jossain vaiheessa tilassa, jossa tiedot ovat tilapäisesti vanhentuneita. (Strauch n.d. 31-32.)

Lopulta oikeellinen engl. Eventual consistency tarkoittaa, että viimeisin syötetty tieto palautetaan sitä kysyttäessä, jos tieto ei ole päivittynyt tietyn ajan kuluessa. Tällä tavoin toimiva järjestelmä on esimerkiksi DNS Internet-nimipalvelu. NoSQL-tietokannoissa tämä käsite liittyy siihen, kuinka suuressa hajautetussa tietokannassa tiedot päivittyvät eri solmukoneiden välillä. Järjestelmä toimii nopeammin ja tehokkaammin, kun yhdessä solmussa päivittyviä tietoja ei viedä heti esimerkiksi 1000:lle muulle saman järjestelmän solmukoneelle. (Vogels 2007.)

CAP-teoreema kertoo, mitä vaatimuksia hajautetulla tietokannalla voi olla. Teoreemassa on kolme termiä, joista vain kaksi voidaan kerralla saavuttaa. Kuvassa 7. on esitetty keltaisella kaksi tekijää, jotka voidaan saavuttaa relaatiotietokannalla ja vihreällä ne kaksi, jotka voidaan saavuttaa NoSQL-tietokannalla. Mustaa keskialuetta eli kaikkia kolmea ei voi saavuttaa. (Harrison. 2013.)

Kuvassa 7 esiintyvät kolme vaatimusta ovat seuraavat:

- Eheys engl. Consistency, tiedot saatavilla jokaisessa solmussa samaan aikaan.
- Saatavuus engl. Availability, tietoa pitää pystyä aina lisäämään ja lukemaan.
- Osituksen sietokyky engl. Partition tolerance, järjestelmä toimii, vaikka kaikki verkon solmut eivät vastaa.



Kuva 7. Cap-teoreema.

Hyvän perustelun NoSQL:n eheysmallille antaa Martin Fowlerin luento, jonka perusteella voidaan ajatella esimerkiksi maailmanlaajuinen verkkokauppa, jossa järjestelmän sovellus- ja tietokantapalvelimet ovat hajautettuina ympäri maailmaa. Jos ajatellaan, että yhteys Euroopan ja Yhdysvaltojen palvelimien välillä on poikki, niin tällöin meillä on kaksi vaihtoehtoa. Ilmoitetaan Euroopan käyttäjille, että verkkokauppa on poissa käytöstä tai annetaan Euroopan palvelimen olla käytössä, vaikka palvelimet eivät kykene synkronoimaan tietoja keskenään. Ensimmäinen vaihtoehto on huono liiketoiminnan kannalta ja jälkimmäisessä on riski päällekkäisiin tietoihin. Kyseessä on valinta, jossa on valittava joko tietojen saatavuus tai eheys. Molempia ei voi saavuttaa yhtäaikaa. Liiketoiminnallisesti saatavuus on tärkeämpää kuin eheys. Riski eheydessä voidaan korjata sovellustasolla tai käyttämällä NoSQL-tietokantaa. NoSQL voi tuoda myös liiketaloudellista hyötyä relaatiotietokantaa heikommalla eheydellä. (Fowler. 2013.)

NoSQL-tietokantojen tapa tallentaa tietoa ja tapa, jolla tietoa luetaan tietokannasta, tekee tiedonmallintamisen erilaiseksi kuin relaatiotietokannoilla. Relaatiotietokannoilla järkevä tapa on jakaa tieto eri tauluihin ja linkittää ne toisiinsa avainkentillä. NoSQL-tietokannat antaa enemmän vapauksia tiedonmallinnukseen, mutta rajoitteena voi olla tiedonhaun tarpeellisuus

yhdellä kyselyllä. Ajatellaan esimerkkinä blogi-järjestelmän tietokantaa, jossa on tekstejä ja niihin kuuluvia kommentteja. Relaatiotietokannalla jokainen teksti kuuluu yhteen tauluun ja kommentit toiseen. Kommenttitauluissa on kenttänä siihen liittyvän tekstin yksilöllinen tunniste, jolloin teksti ja siihen liittyvät kommentit saadaan yhdellä JOIN-kyselyllä. Verkkotietokantoja lukuunottamatta NoSQL-tietokannat eivät tue liitoksia. NoSQL tietokannoissa sama asia voidaan esittää myöskin samalla tavalla kuin relaatiotietokannoissa, mutta silloin tietoa haettaessa tulee käyttää kaksi hakua. Parempi ratkaisu on käyttää sellaista objektimuotoista esitystä, jossa kommentit ovat objektitaulukkona tekstiobjektin sisällä, kuten esitetään allaolevassa esimerkissä 1, jossa teksti avaimen sisältö on itse blogiteksti ja toinen avain kommentit sisältää taulukon kommentti-objekteja. (Katsov 2012.)

```
{ ”teksti”: ”yksi kaksi kolme...” , ”kommentit”:[
  {”kommentti”:”neljä viisi....”},
  {”kommentti”:”kuusi seitsemän....”}, ...
]}
```

Esimerkki 1. Tietomalli NoSQL-tietokannassa.

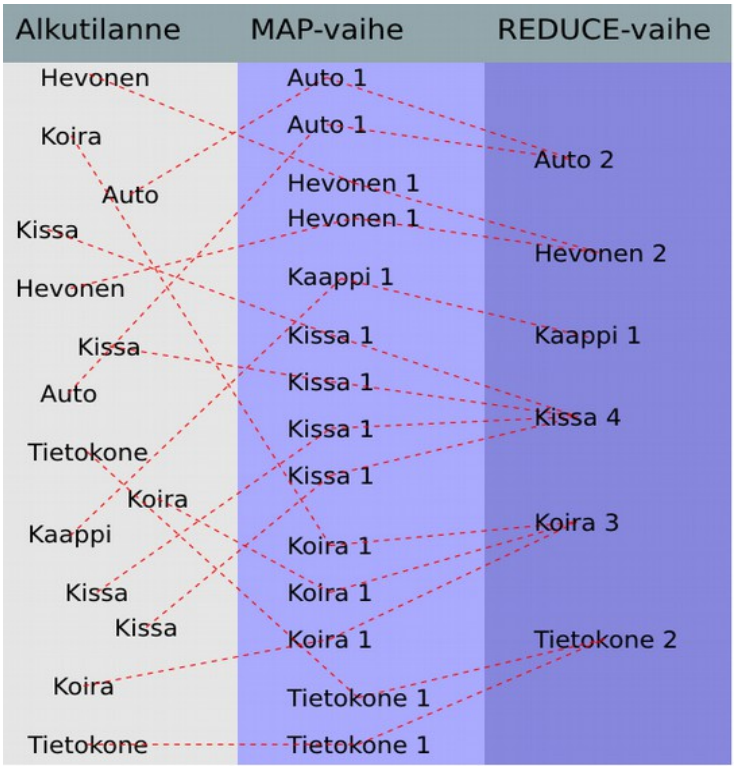
Etuna tässä mallissa on, että kaikki tieto on haettavissa yhdellä kyselyllä ja yleensä rakenne saadaan ohjelmointikielissä suoraan oikeaan rakenteeseen esimerkiksi JSON-objektiksi Node.js:ssä. Tämä malli ei ole ainoa mahdollinen, vaan se riippuu tiedon rakenteesta ja siitä, miten tietoa tullaan hakemaan ja käsittelemään.

3.4 Tiedonkäsittely Map/Reducella

Tietojen hajautus usealle palvelimelle luo tilanteen, jossa perinteinen tietojen kysely SQL-kielellä ei ole mahdollista. Kyselyjen tekemiseen tarvitaan Googlen kehittämä tekniikka Map/Reduce. Tämä tekniikka on hyvin tyypillinen NoSQL-tietokantaratkaisuissa ja erilaisten Bigdata-teknologioiden yhteydessä. Tämän tekniikan avulla tehtävä jaetaan pienempiin osiin suoritettavaksi usealle solmukoneelle. Ensimmäinen vaihe on aina Map, jossa tieto kootaan ja järjestellään. Map-vaiheen tulokset annetaan syötteenä Reduce-vaiheelle, jonka tehtävänä on suorittaa tiedon koostaminen. Yleisesti voidaan ajatella, että Map-vaihe vastaa SQL-kyselyssä select- ja order by-lauseita. Vastaavasti Reduce-vaihe vastaa group by- ja having-lauseita. Molemmat vaiheet sovelluskehittäjä kirjoittaa itse jollain ohjelmointikielellä, jossa molemmat vaiheet ovat omina funktioinaan. (Nurse, 2013.)

Ajatellaan yksinkertaisena esimerkkinä sanoja, joista halutaan selvittää yksittäisten sanojen lukumäärät yhteensä ja esittää sanat ja niiden lukumäärät aakkosjärjestyksessä. Kuvassa 8 on alkutilanteessa epämääräinen sekalainen joukko sanoja. Map-vaiheessa sanat järjestetään aakkosjärjestykseen

ja niille annetaan lukumääränumero. Viimeisessä Reduce-vaiheessa luvut summataan sanoittain yhteen.



Kuva 8. Esimerkin Map/Reduce

Esimerkistä voidaan helposti havaita, että sekä Map- ja Reduce-vaiheet voidaan jakaa edelleen pienempiin osatehtäviin. Ohjelmoijan ei kuitenkaan tarvitse itse huolehtia siitä, miten vaihe hajautetaan eri solmukoneiden tehtäväksi vaan tietokantajärjestelmä huolehtii siitä automaattisesti. Ohjelmoija voi keskittyä pelkästään ongelman ratkaisuun. Ohjelmoija voi itse päättää, mitä Map- tai Reduce-vaiheessa tiedolle tehdään. Ei tarvitse rajoittua pelkästään tietokannan kyselykielen tarjoamiin mahdollisuuksiin, vaan saadaan vapaus toteuttaa monimutkaisempiakin kyselyjä.

3.5 Riak

Riak on skaalautuva avoimen lähdekoodin avain-arvo-tietokanta ja se perustuu paljolti Amazonin Dynamo-tietokantaan. Riak APIna käytetään Erlang-kielellä toteutettua REST-rajapintaa tai Protocoll buffers rajapintaa. Riakille löytyy sovelluskirjastot yleisimmille ohjelmointikielille. Riak on saatavilla kolmena eri versiona ja avoimen perusversion lisäksi on kaupallinen Enterprise, joka tarjoaa moni-datakeskus replikoinnin ja monitorointityökaluja. Kolmas vaihtoehto on Riak CS, joka on tarkoitettu Amazon-pilvipalvelulle. Riak on tarjolla vain UNIX-pohjaisille käyttöjärjestelmille eli mm. Linuxille. Riakissa ei ole oletuksena hallinta- tai konfigurointisovellusta, mutta yksinkertainen selainkäyttöinen klusterin hallintakonsoli voidaan kytkeä päälle. Riakin konfigurointi suoritetaan aina komentorivi-

työkaluilla, tiedostoja muokkaamalla tai käyttämällä REST-rajapintaa. (Riak 2013 theory/concepts.)

Riakissa avain-arvo-tiedot talletetaan bucketeihin, joiden voidaan ajatella vastaavan tauluja relaatiotietokannassa. Hajautetussa usean solmukoneen Riak-klusterissa ei ole yksittäistä pääsolmukonetta, vaan klusterin kaikki solmut ovat yhdenvertaisia. Riak klusteria kutsutaan kehäksi engl. Ring, joka jaetaan partitioiksi tai virtuaalisolmuiksi. Kehän virtuaalisolmut jaetaan edelleen fyysisille solmukoneille. Fyysisiä solmukoneita voidaan lisätä tai poistaa dynaamisesti tietokantaa sammuttamatta. (Riak 2013 theory/concepts.)

Riakin oletustallennusmoottori on Bitcask, mutta rakenteen vuoksi se voidaan vaihtaa. Bitcask on yksinkertainen, mutta tehokas. Bitcaskissa avaimet tallennetaan käyttömuistiin nopeuttamaan tiedon noutoa. Toinen yleisesti käytetty tallennusmoottori on LevelDB, joka antaa mahdollisuuden luoda toisio-indeksejä engl. Secondary index. (Riak 2013 theory/concepts.)

3.5.1 HTTP-rajapinta

Riakin tiedon tallennus- ja lukuoperaatiot tehdään REST-rajapinnan avulla HTTP-protokollalla. Tiedon vienti tapahtuu käyttämällä PUT- tai POST-pyyntömetodia. Kuvassa 9. tallennetaan paikallisen koneen eli localhostin Riak-tietokantaan esimerkki-nimiseen buckettiin JSON-dokumentti käyttämällä avainta henkilo1. Pyynnön parametrinä kerrotaan, että pyynnön runko eli tallennettu tieto palautetaan. Jos sama komento annetaan uudestaan niin, tieto kirjoittuu edellisen päälle, koska avain on sama. Jos halutaan, että Riak muodostaa avaimen automaattisesti, niin pitää käyttää POSTia ja jättää avain pois. (Riak 2013 dev/references/http.)

```
kl@riakserver1:~$ curl -v -XPUT http://127.0.0.1:8098/riak/esimerkki/henkilo1?returnbody=true -H "Content-Type
: application/json" -d '{"nimi": "Kari Lehtomaa"}'
* About to connect() to 127.0.0.1 port 8098 (#0)
*   Trying 127.0.0.1... connected
> PUT /riak/esimerkki/henkilo1?returnbody=true HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/
2.3
> Host: 127.0.0.1:8098
> Accept: */*
> Content-Type: application/json
> Content-Length: 25
>
* upload completely sent off: 25out of 25 bytes
< HTTP/1.1 200 OK
< X-Riak-Vclock: a85hYGBgzGDKBVIcypz/fgbu/1eYwZTIlMfKYJvgf5ovCwA=
< Vary: Accept-Encoding
< Server: MochiWeb/1.1 WebMachine/1.9.2 (someone had painted it blue)
< Link: </riak/esimerkki>; rel="up"
< Last-Modified: Mon, 08 Jul 2013 16:09:33 GMT
< ETag: "6wZMy9A7QpX25CWLOM1XZA"
< Date: Mon, 08 Jul 2013 16:09:33 GMT
< Content-Type: application/json
< Content-Length: 25
<
* Connection #0 to host 127.0.0.1 left intact
* Closing connection #0
{"nimi": "Kari Lehtomaa"}kl@riakserver1:~$
```

Kuva 9. Tiedon tallennus Riakiin Curlilla.

Tietoja haettaessa käytetään GET-pyyntöä ja jos pyyntöön annetaan avain, niin kyseinen avaimen tieto noudetaan, kuten kuvassa 10. Ilman avainta ja parametrillä keys=true, palautetaan kaikki avaimet kyseisessä bucketissa. Tallennuksen ja noudon lisäksi tietoa voidaan poistaa DELETE-pyynnöllä.

```
kl@riakserver1:~$ curl -v http://127.0.0.1:8098/riak/esimerkki/henkilo1
* About to connect() to 127.0.0.1 port 8098 (#0)
*   Trying 127.0.0.1... connected
> GET /riak/esimerkki/henkilo1 HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/
2.3
> Host: 127.0.0.1:8098
> Accept: */*
>
< HTTP/1.1 200 OK
< X-Riak-Vclock: a85hYGBgzGDKBVIcypz/fgbu/1eYwZTIlMfKYJvgf5ovCwA=
< Vary: Accept-Encoding
< Server: MochiWeb/1.1 WebMachine/1.9.2 (someone had painted it blue)
< Link: </riak/esimerkki>; rel="up"
< Last-Modified: Mon, 08 Jul 2013 16:09:33 GMT
< ETag: "6wZMy9A7QpX25CWLOM1XZA"
< Date: Mon, 08 Jul 2013 16:32:05 GMT
< Content-Type: application/json
< Content-Length: 25
<
* Connection #0 to host 127.0.0.1 left intact
* Closing connection #0
{"nimi": "Kari Lehtomaa"}kl@riakserver1:~$
```

Kuva 10. Tiedon haku avaimella.

3.5.2 Hakutoiminnot

Riakissa tietoa voidaan hakea kolmella eri tavalla: suoraan avaimella, Map/Reduce-työllä tai hakutoiminnolla. Map/Reducen avulla voidaan suorittaa hakuja, suodatuksia ja laskentaa jakamalla työ useammalle solmukoneelle. Työt voidaan kirjoittaa Javaskriptillä tai Erlangilla ja työ lähetettään Riakiin HTTP APIn tai Protocoll buffersin kautta. Riakin Map/Reduce on tarkoitettu ajettavaksi jonotyönä eikä tosiaikakyselynä. Ennen Map-vaihetta käsiteltäviä tietoja voidaan suodattaa käyttämällä avainfilttereitä.

Kuvassa 11 suoritetaan kuvan 8 mukainen Map/Reduce käyttämällä Curlia. Itse Map/Reduce koodi ladataan count.json-tiedostosta, joka on esitetty esimerkissä 2. Eli map-vaiheessa noudetaan arvot, jotka muodostetaan JSON-objekteiksi siten, että kentän nimi on noudettu arvo ja kentän arvo on 1. Reduce-vaiheessa suoritetaan yhteenlasku.

```
kl@riakserver1:~/mapred$
kl@riakserver1:~/mapred$ curl -v http://localhost:8098/mapred -H 'Content-Type: application/j
son' -d @count.json
* About to connect() to localhost port 8098 (#0)
*   Trying 127.0.0.1... connected
> POST /mapred HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 lib
idn/1.23 librtmp/2.3
> Host: localhost:8098
> Accept: */*
> Content-Type: application/json
> Content-Length: 451
>
* upload completely sent off: 451out of 451 bytes
< HTTP/1.1 200 OK
< Server: MochiWeb/1.1 WebMachine/1.9.2 (someone had painted it blue)
< Date: Thu, 18 Jul 2013 13:45:49 GMT
< Content-Type: application/json
< Content-Length: 69
<
* Connection #0 to host localhost left intact
* Closing connection #0
[{"Koira":3,"Kissa":4,"Auto":2,"Hevonen":2,"Kaappi":1,"Tietokone":2}]kl@riakserver1:~/mapred$

kl@riakserver1:~/mapred$
kl@riakserver1:~/mapred$
```

Kuva 11. Suoritetaan Map/Reduce HTTP:llä.

```
{"inputs":"elaimet",
  "query":[{"map":{"language":"javascript",
            "source":"
            function(v){
                        var m = v.values[0].data;
                        var r = [];

                        if(m != '') {
                                    var o = {};
                                    o[m] = 1;
                                    r.push(o);
                        }
                          return r;
            }
"}},
{"reduce":{"language":"javascript","source":"
            function(v) {
             var r = {};
              for(var i in v) {
                for(var w in v[i]) {
                  if(w in r) r[w] += v[i][w];
                  else r[w] = v[i][w];
                }
              }
              return [r];
            }
"}}
          ]
}
```

Esimerkki 2. Count.json Map/Reduce koodi

Riakin hakutoiminto on full-text hakumoottori, joka tulee Riakin mukana. Hakutoiminto pitää kytkeä päälle konfiguraatiotiedostosta ja se vaatii vielä bucket-kohtaisen aktivoinnin ennen tietojen tallennusta. Haku tukee erilaisia tietokantaan tallennettuja tietomuotoja, kuten mm. JSON, XML ja teksti. Hakurajapinta käyttää samaa syntaksia kuin Lucene ja tukee useimpia Lucenen operaattoreita. Haun tulokset voidaan ohjata Map/Reduceen jatkokäsittelyyn. Haku voidaan suorittaa käyttämällä komentorivityökalua tai HTTP:n kautta. Allaolevassa esimerkissä 3 on esitetty erilaisia haku mahdollisuuksia. Ensimmäisessä haetaan tietueita, joiden nimi-kentässä on teksti Kari. Toisessa haetaan tietueita, joiden nimi kentän tieto alkaa joko merkkijonolla Ka tai Pe. Kolmannessa haetaan tietueet, joiden ika-kentän arvo on 30:n ja 40:n välillä. Haun käyttämistä rajoittaa se, että sanasta tulee antaa 2 ensimmäistä kirjainta. Hakumoottoria varten kentille muodostetaan oletusskeema, mutta skeemaa voi muokata halutuksi. Esimerkiksi oletuksena isot ja pienet kirjaimet ovat eri asia.

```
Nimi:Kari
Nimi:Ka* OR Nimi:Pe*
Ika:[30 TO 40]
```

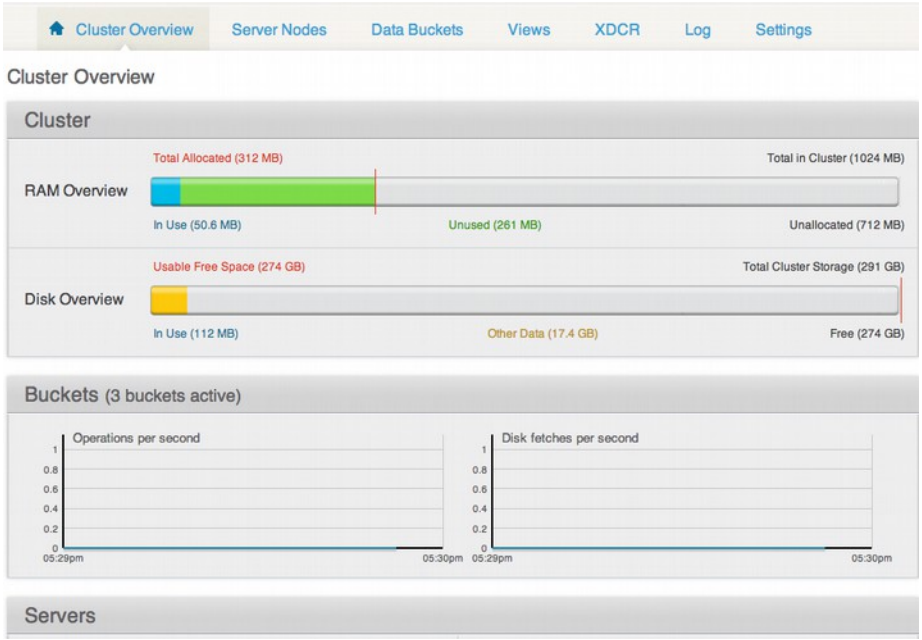
Esimerkki 3. Riak-haku mahdollisuuksia.

3.6 Couchbase

Couchbase on dokumenttipohjainen tietokanta, johon dokumentit määritellään JSON-muotoisina. Couchbase on saatavilla Enterprise tai Community versioina. Molemmat voidaan ladata ilmaiseksi Linuxille, Macille ja Windowsille ja lisäksi Enterprise-versioon on myös saatavilla kaupallista tukea.

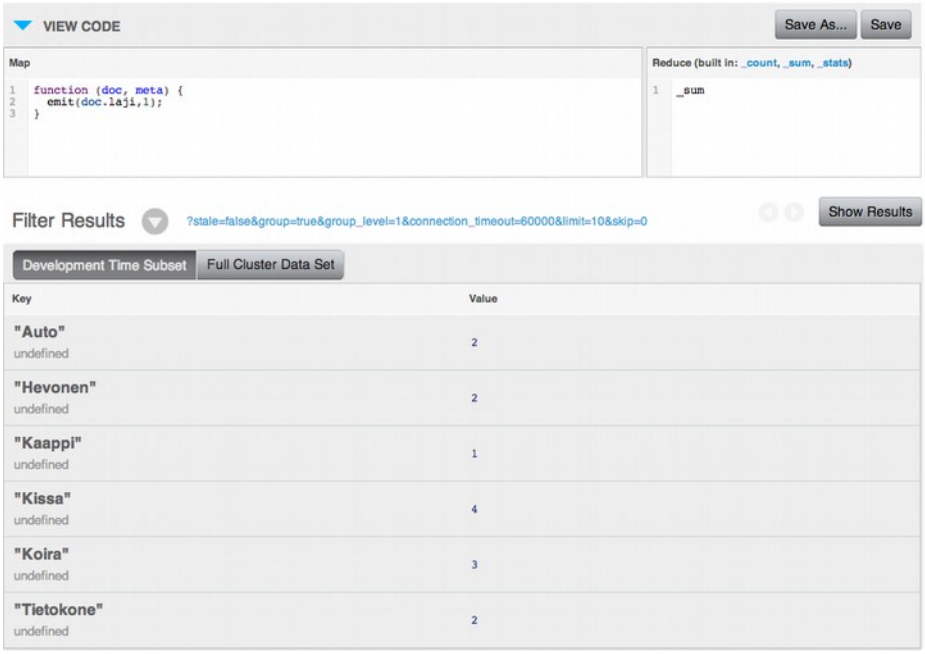
Couchbase on yhdistelmä Membase- ja Couchdb-tietokannoista. Couchbase tallentaa myös tiedot bucketeihin, mutta erona Riakiin on, että bucket voi olla kahta eri tyyppiä. Couchbase tallentaa tiedot dokumenttimuotoisina levylle ja käyttää memcachedia välimuistina ja avainten säilytykseen. Memcached-tyyppisessä bucketissa tiedot ovat avain-arvo tyyppisiä ja ne tallennetaan ainoastaan muistiin. Dokumenteille voidaan määritellä myös voimassaoloaika. Couchbase voidaan hajauttaa horisontaalisti kappaleen 3.1 mukaan, jolloin ympäristössä ei ole erillistä keskuskonetta vaan jokainen solmu tarjoaa saman palvelun. Couchbasen hajautus käyttää vBucket-tekniikkaa, jossa dokumenttien avaimet voidaan kohdistaa niitä säilöviin solmukoneisiin. Couchbasessa on hajautuksen lisäksi tietojen replikointi, jolla bucket voidaan kopioida toiseen. Couchbaselle on REST-rajapinta ja sille löytyy myös sovellusrajapinnat yleisimmille ohjelmointikielille. Couchbasen REST-rajapinta toimii hyvin samalla tavoin kuin Riakissa. (Couchbase 2013.)

Kuvassa 12 on Couchbasen selainpohjainen hallintakonsoli, jonka kautta voidaan hoitaa kaikki ylläpitotoimet. Hallintakonsolin kautta voidaan suorittaa myös dokumenttien ja näkymien syöttö ja muokkaus.



Kuva 12. Couchbase-hallintakonsoli, etusivu.

Couchbasessa ainoa keino suorittaa kyselyitä pelkän avaimella noudon sijaan on näkymien käyttäminen. Näkymät laaditaan Javaskriptillä ja niihin luodaan Map- ja Reduce-funktiot. Näkymät voivat olla joko kehitys- tai tuotantonäkymiä. Näkymät ovat itsessään tietynlaisia JSON-dokumenttejä bucketissa, jolloin niitä voidaan luoda Couchbasen rajapintojen kautta. Näkymää käytettäessä sille voidaan määritellä tulosjoukkoa tarkentavia tietoja filttereillä, esimerkiksi alku- ja loppuavain tai jokin tietty avain tai avainjoukko. Reduce-vaihetta varten voidaan määritellä, että tiedot ryhmitellään SQL:n group by-tyylisesti. Kuvassa 13 esitetään näkymämäärittely ja tulokset Couchbasen hallintakonsolissa. Esimerkkinä on kuvan 8 mukainen tietojoukko, jossa dokumentin kenttänä on laji. Map-vaihe lähettää Reduce-vaiheelle jokaisen dokumentin lajin ja numeron 1. Jotta tiedot näkyvät ryhmiteltynä, tulee tulosjoukon filtterinä olla group valittuna ja group_levelin arvona 1. Kun koodia verrataan Riakin vastaavaan esimerkissä 2, voidaan huomata, että Map/Reduce-toteutus Couchbasessa on huomattavasti yksinkertaisempi kuin Riakissa.



Kuva 13. Esimerkkinäkymä Couchbasessa.

4 OHJELMOINTIYMPÄRISTÖ NODE.JS

Node.js on Googlen V8-javaskriptimoottorin päälle rakennettu suoritusalusta, jolla on helppo kehittää nopeita ja skaalautuvia verkkopalveluita. Node.js on javaskriptiä, jota suoritetaan palvelimella. Node.js on tapahtumapohjainen, engl. Event-driven ja se käyttää asynkronista eli keskeyttämätöntä I/O mallia. Asynkronisuus on Node.js:n keskeisin ja hyödyllisin ominaisuus. (Node.js n.d.)

Kun ohjelmakoodissa käytetään I/O:ta eli suoritetaan luku- tai kirjoitusoperaatioita tiedostoon, tietokantaan tai verkkoon, kyseisen prosessin suoritus jää odottamaan toiminnon päättymistä. Prosessi voidaan jakaa säikeisiin, jolloin yksi säie voi jäädä odottamaan ja muut voivat jatkaa suoritusta. Monisäikeisessä kehitysmallissa ohjelmoija joutuu kuitenkin tekemään monimutkaisia tarkistuksia ja rakenteita. Javaskriptin tapahtumapohjaisessa ohjelmoinnissa koodia suoritetaan asynkronisesti tapahtumien perusteella. Tällaisessa ohjelmoinnissa ei yleensä käytetä funktioiden palautusarvoja, vaan ohjelmoija määrittelee takaisinkutsu-funktion engl. Callback-function, jota kutsutaan operaation tapahtuman yhteydessä. Kun takaisinkutsu-funktiota suoritetaan, niin se muistaa ympäristön, missä se määriteltiin eli etukäteen määritellyt muuttujat ovat käytettävissä. Tällaista menetelmää sanotaan closureksi. (Teixeira 2013 15–18.)

Allaolevassa esimerkissä 4. esitetään perinteisen mallin mukainen ohjelman suoritus. Ohjelman prosessi jää odottamaan riville 2 TiedostoTallenna- funktion päättymistä. Esimerkissä 5 on sama toiminto toteutettuna Node.js:llä ja selkeänä erona on rivillä 2 funktion sisällä oleva funktio määritys: function(err). Tämä funktio on asynkroniseen toimintaan kuuluva takaisinkutsu-funktio. Takaisinkutsu-funktiot voivat olla nimettömiä, kuten tässä esimerkissä tai ne voivat olla nimettyjä ja muualla koodissa määriteltyjä. Prosessi ei jää odottamaan tiedostoon tallennus-funktion loppumista, jolloin ohjelma tulostaa esimerkin 6. mukaan.

```
1. Tulosta(”Kirjoitetaan tiedostoon”)
2. TiedostoTallenna(”Tiedostonnimi”,”Terve”)
3. JOS VIRHE
4.         Tulosta(”Virhe tallennuksessa”)
5. MUUTEN
6.         Tulosta(”Tallennus valmis”)
```

Esimerkki 4. Pseudokoodi esitys perinteisestä ohjelmointimallista.

```
1. console.log(”Kirjoitetaan tiedostoon”);
2. fs.writeFile('Tiedostonnimi', 'Terve', function (err) {
3. if(err) {
4.         console.log(”Virhe tallennuksessa”);
5.         }
6. else { console.log(”tallennus onnistui”); }
7. });
8. console.log(”Milloin tämä tulostuu?”);
```

Esimerkki 5. Node.js javaskriptikoodi tiedoston tallennuksesta.

```
Kirjoitetaan tiedostoon

Milloin tämä tulostuu?

tallennus onnistui
```

Esimerkki 6. Edellisen koodin tuloste.

Asynkroninen ohjelmointi tekee ohjelman suorituksesta nopeampaa, mutta vaarana voi olla ohjelmakoodin luettavuuden huonontuminen. Tällainen ohjelmointityyli voi tehdä koodista sekavaa ja aina asynkroninen ohjelmointi ei ole mahdollista. Node.js:ään on saatavilla lisäkirjastoja, joilla asioita voidaan pakottaa tekemään tietyssä järjestyksessä. Javaskripti ei myöskään ole oikea olio-ohjelmointikieli vaikka siinä voidaan luoda objekteja.

4.1 Tiedon kuvausmuoto JSON

Node.js käyttää tietomuotona JSONia. JSON tulee englanninkielen sanoista JavaScript Object Notation. Se on kevytrakenteinen tekstimuotoinen tiedon kuvausmuoto ja kuuluu osana javaskriptin standardiin. Se on helppolukuista niin ihmisille kuin ohjelmille. XML:ään verrattuna sen käsittely ohjelmallisesti on huomattavasti yksinkertaisempaa. JSON ei myöskään riipu käytettävästä ohjelmointikielestä. JSON on erittäin hyödyllinen varsinkin AJAX-pyyntöjen tiedonvälityksessä. (JSON n.d.)

JSON-objekti rakentuu nimi-arvo-pareista tai taulukosta arvoja. Arvo voi olla taulukko tai se voi olla JSON-objekti. Totuusarvo voidaan esittää lauseilla true tai false. Objektin määritys sijoitetaan kaarisulkujen sisään, kun kyseessä on nimi-arvo yhdistelmä tai hakasulkujen sisään, kun kyseessä on taulukko. Seuraavassa esimerkissä 7 on esitelty erilaisia vaihtoehtoja.

```
{ ”etunimi”:”Kari”, ”sukunimi”:”Lehtomaa” }
[”Kari Lehtomaa”, ”Tero Testaa”]
{ ”Tietue”:{”nimi”: ”Kari Lehtomaa”}}
{ ”Tietue”:[”Kari Lehtomaa”, ”Tero Testaa”]}
{”TilaValmis”: true}
```

Esimerkki 7. Erilaisia JSON-dokumenttejä

Node.js:ssä JSON-tiedosto ladataan require-lauseella:

```
var objekti = require('./esim.json');
```

Nimi-arvo pareihin voidaan viitata kahdella eri tavalla, kun halutaan käsittellä arvoa:

```
objekti['etunimi'] tai objekti.etunimi
```

Ja taulukoihin []-operaattorilla:

```
objekti[0]
```

4.2 Lisäkirjastot

Javaskriptin toimintatapaan kuuluu, että kaikki koodi on samassa nimiavaruudessa. Saman nimiavaruuden käyttäminen aiheuttaa helposti vaikeasti selvitettäviä virheitä, koska ohjelmoija saattaa käyttää esimerkiksi samoja funktioiden nimiä eri tiedostoissa. Lisäksi malli voi aiheuttaa tietoturva-aukkoja. Node.js:ssä ongelma voidaan välttää käyttämällä ladattavia lisäkirjastoja, joissa ohjelmakoodi on omassa kontekstissaan. Tämä tekee koodista helpommin hallittavaa. Kirjastoja voi tehdä itse ja niitä tulee paljon Node.js:n mukana tai niitä voi asentaa myös lisää npm-komentorivityökalun avulla. Sivustolla npmjs.org on nähtävillä lisäkirjastot, jotka voidaan asentaa suoraan npm-työkalulla komennolla: npm install kirjastonnimi. Esimerkissä 8 on esitetty kuinka luodaan oma lisäkirjasto ja kuinka se ladataan omassa koodissa. (Teixeira 2013 23-24.)

```
// lisa1.js
function kirjoitaTeksti (teksti){
    console.log(teksti);
}

module.exports = kirjoitaTeksti;

// ohjelma.js
var oma1 = require('./lisa1.js');
oma1.kirjoitaTeksti(”Moi maailma”);
```

Esimerkki 8. Oma lisäkirjasto ja sen käyttäminen.

4.3 Verkko-ohjelmointi

Verkkopalvelu tarvitsee aina HTTP-palvelimen ja yleisin tällainen on Apache. Node.js:llä toteutettua sovellusta ei voi kuitenkaan suorittaa minkään valmiin palvelimen päällä, vaan palvelin pitää toteuttaa sovelluksen sisään. Node.js sisältää valmiit peruskirjastot, joilla palvelimen luonti onnistuu. Tuki ei rajoitu pelkästään HTTP:hen, vaan Node.js:llä voidaan toteuttaa myös muita protokollia käyttäviä palveluita. Allaolevassa esimerkissä 9 on yksinkertainen koodi HTTP-palvelimesta, joka kuuntelee pyyntöjä osoitteessa http://localhost:8080. Skripti näyttää selaimessa tekstin Moi maailma otsikkona, kun skripti on käynnistetty palvelimella komennolla: node palvelin.js. (Teixeira 2013 95-96.)

```
// palvelin.js
var webbi = require('http');
webbi.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('<h1>Moi maailma</h1>\n');
}).listen(8080, '127.0.0.1');

console.log('Palvelin http://127.0.0.1:8080/');
```

Esimerkki 9. Yksinkertainen HTTP-palvelin Node.js:llä.

Yleensä sovelluksissa käytetään sopivaa lisäkirjastoa, joiden avulla ohjelmoija voi keskittyä enemmän itse verkkosovelluksen tekemiseen. Hyvin paljon käytetty tällainen lisäkirjasto on Express, johon on saatavilla myös erilaisia HTML-sivumallimoottoreita, joiden avulla dynaamisten HTML-sivujen käsittely onnistuu helposti. Liitteessä 1 on esitetty peruspohja Node.js Express-lisäkirjastolla toteutettavaa verkkosovellusta varten. Peruspohjassa on mallina yksi pyyntöreitti, engl. route:

```
app.get('/', function(...
```

Kyseinen rivi määrittelee sen, että palvelu vastaa pyyntöön http://palvelin/. Funktion sisällä on komennot res.local, joka määrittelee sivupohjalle lähetettävät tiedot ja res.render, joka huolehtii sivun lähettämisestä takaisin selaimelle. Pyyntöreitin avulla palveluun saadaan toteutettua kaikki REST-pyynnöt. Pyyntöreittinä voi olla esimerkiksi myös POST-muotoinen pyyntö, joka vastaa pyyntöön http://palvelin/palvelu:

```
app.post('/palvelu', function(...
```

Palvelimen lisäksi Node.js:llä voidaan toteuttaa HTTP-asiakassovellus. Sen voi toteuttaa ympäristön vakiokirjaston avulla, mutta helpompi tapa on käyttää Mikeal Rogersin request-kirjastoa. Esimerkissä 10 on esitetty yksinkertainen HTTP-asiakasohjelma, joka tulostaa sivun www.yle.fi HTML:n. Pyynnön toimintaa ohjataan asettamalla options-objektiin haluttuja arvoja. Request-kirjastolla voidaan toteuttaa myös kaikki REST-pyynnöt. Ennen kuin skripti toimii, pitää Request-kirjasto asentaa skriptin hakemistoon komennolla: npm install request. Huomioitavaa request-funktiossa on se, että sen suoritus päättyy , mutta sen sisällä määritelty asynkroninen funktio jatkaa toimintaa, kunnes osoitteen mukainen sisältö on ladattu kokonaan. (Teixeira 2013 118-127.)

```
var request = require('request');

var options = { url: "http://www.yle.fi", method: "GET" };
  request(options, function(error, response, body){
    console.log(body);
});
```

Esimerkki 10. Yksinkertainen HTTP-asiakas.Node.js:llä.

Node.js:n asynkronisesta toiminnasta hyödytään nimenomaan verkkosovellusten kanssa, koska perinteisessä mallissa jokainen yhteys verkkopalvelimelle avaa uuden säikeen. Jokaisella uudella säikeellä on oma resurssipooli, johon kuuluvat esimerkiksi muistialue ja avatut tiedostot. Näin toimii esimerkiksi Apache-palvelin. Node.js ei käytä säikeitä vaan pyyntöihin vastataan tapahtumapohjaisesti ja asynkronisesti. Tällöin sovelluksista saadaan nopeita ja ne vievät vähemmän muistia. Ongelmana Node.js:n tavassa toimia on mahdollisuus siihen, että yhteen pyyntöön vastaaminen

kestää liian kauan, jolloin se saattaa jumittaa koko palvelun. Toinen ongelma on se, että sovellus kaatuu virheeseen tai virhe aiheuttaa poikkeuksen. Täten tuotantokäytössä oleva Node.js:llä toteutettu verkkopalvelu vaatii lisätoiminnon, jolla estetään ajonaikaisten virheiden aiheuttama sovelluksen kaatuminen. (Capan 2013.)

5 TOIMINTOJEN SUUNNITTELU JA TOTEUTUS

Ennen toteutuksen aloitusta kehitystyötä varten määriteltiin ja rakennettiin kehitysympäristö. Kehitysympäristö määriteltiin tulevan mahdollisen tuotantoympäristön mukaan ja tällöin käyttöjärjestelmäksi valittiin 64-bittinen Ubuntu Linux versio 12.04 LTS. Kehitysympäristö asennettiin Virtualbox virtuaalikoneeseen käyttämällä kevyempää Xubuntua, joka eroaa vakioversiosta vain keveämmän käyttöliittymän osalta. Tietokantojen osalta asennettiin Riak versio 1.3.1 ja Couchbase 2.1.1. Node.js versiona oli 0.10.10. Pääasiallisina kehitystyökaluina olivat Git-versiohallinta ja Geany tekstieditori. Couchbasea lukuunottamatta kaikki tarvittavat ohjelmistot asennettiin Ubuntun normaalilla pakettienhallinta-ohjelmistolla.

Toimintojen suunnittelua varten oli käytettävissä toimeksiantajan kanssa laadittu kevyt määrittely vaadituista toiminnoista sekä Editan toimittamat määrittelyt ja XSD-skeemat. Kehitystyön muutokset hallittiin Git-versionhallinnalla ja muutokset vietiin aina Githubiin. Toteutuksen aikana toimeksiantajan kanssa pidettiin viikottain lyhyt palaveri, jossa käytiin läpi tehtyjä asioita ja tehtiin tarvittaessa muutoksia määrittelyyn. Lähtökohtana toteutukselle käytettiin myös Pythonilla tehtyä vanhaa noutosovellusta sekä siihen asti tehtyä hankintailmoitussovellusta. Uusi hankintailmoitusportaali oli toteutettu Node.js-alustalle ja se käytti ulkoasukirjastona Twitter bootstrappia ja selainpuolen toiminnoissa käytettiin Javaskriptiä, Jqueryä ja Jquery UI:tä.

5.1 Git-versionhallinta

Sekä alkuperäinen noutotoiminto että Hilma-sovellus olivat saatavilla toimeksiantajan yksityisistä Github-versionhallintavarastoista. Github on eräänlainen pilvipalvelu, johon kuka tahansa voi rekisteröityä ja luoda omia Git-varastoja. Github mahdollistaa yhdessä työskentelyn paikasta ja ajasta riippumatta. Ilmaisella käyttäjätilillä voi luoda vain julkisia varastoja, kun taas maksullisella onnistuu myös yksityisten varastojen luonti. Githubissa voidaan myös selata ohjelmakoodia selaimella.

Yleisesti versionhallinta tarkoittaa sitä, että järjestelmä tallentaa muutoshistoriaa tiedostoista, jotka on liitetty versionhallintaan. Versionhallinta mahdollistaa, että sieltä voidaan ladata haluttu aiempi versio. Versionhallinnan paikkaa, johon historia talletetaan, sanotaan varastoksi, engl. repo-

sitory. Git on jaettu engl. distributed versionhallintaohjelmisto, jossa varasto voi sijaita etäpalvelimella tai paikallisella koneella. (Vogel 2013.)

Git-varasto voi luoda haluttuun hakemistoon komennolla git init. Yleisesti sovelluskehitys Git:llä Githubin kanssa menee seuraavasti:

```
1. git clone http://github/projekti
2. cd projekti
3. Muokkaa tiedostoja
4. git commit -a -m 'Tehty muutos....'
5. git push origin
6. git pull origin
```

Ensin kloonataan Github-projekti omalle työasemalle ja siirrytään työhakemistoon. Kohdassa 3 – 5 suoritetaan silmukkana toimenpiteet: tiedostojen muokkaus, muutosten tallennus omaan Git-varastoon ja push-komennolla viedään muutos Githubiin. Kohdissa 5 ja 6 origin tarkoittaa sitä varastoa, josta projekti on noudettu git clone–komennolla. Kohdan 6. komennolla voidaan ladata omaan varastoon muiden käyttäjien tekemät muutokset. Jos työhakemistoon lisätään uusia tiedostoja tai hakemistoja, niin ne tulee lisätä komennolla git add tiedostonimi. Suositeltavaa on, että 3 – 5 kohtien ketju olisi lyhyt ja sisältäisi vain yhden muutoksen kerrallaan.

5.2 Tietokanta

Sekä Riak että Couchbase valittiin ehdokkaiksi sen perusteella, että ne käyttävät tiedon siirtomuotona JSONia. Riak oli suunnitteilla myös toimeksiantajan toiseen projektiin, jolloin tämän työn mukana saataisiin kokemusta ja osaamista Riakin käytöstä. Erittäin olennainen tekninen vaatimus valittavalle tietokannalle oli se, että siihen on olemassa lisäkirjastot Node.js:lle. Sekä Riakille että Couchbaselle löytyy toimivat lisäkirjastot npmjs.org-sivustolta. Molemmissa havaittiin hyvin aikaisessa vaiheessa, että lisäkirjastojen ohjeistus oli hyvin puutteellista, jolloin lisäkirjastojen tarjoamien funktioiden toimintaa ja käyttöä piti tutkia lisäkirjastojen lähdekoodista saakka. Molemmat tietokannat tukevat JSON:ia tallennusmuotona, jolloin voidaan käyttää suoraan Node.js:n käyttämää tietomuotoa ilman konvertointia. Molemmissa tietokannoissa on myös tarvittavat työkalut ja toiminnot tiedonhakuun. Riakissa on Map/Reducen lisäksi vielä oma hakumoottori.

Tietokantoja testattiin käytännössä tarkemmin noutotoiminnon toteutuksen yhteydessä. Molempien tietokantojen kohdalla tietokantayhteyden luonti ja tiedon tallennus onnistui lisäkirjastojen avulla. Kuitenkin Couchbasen yhteydessä havaittiin ongelma, johon ei löytynyt ratkaisua. Suoritettaessa

Couchbasen tallennus-funktiota ohjelma jää odottamaan seuraavaa tallennusta ja siihen ei löytynyt ratkaisua, jolla se olisi voitu päättää, kun ohjelman toiminta päättyy. Toiminta ei ole virheellistä, koska verkkosovelluksessa se toimii kuten pitääkin. Noutosovellus ei ole verkkosovellus vaan ajettava skripti, jonka tulee päättyä, kun kaikki tieto on käsitelty.

Liitteessä 2 on ohjelmakoodi, jolla tiedontallennus tehdään Node.js:ssä Couchbaseen ja vastaava koodi Riakia varten on liitteessä 3. Molemmat tietokannat toimivat tallennuksessa siten, että olemassaoleva tieto korvautuu, jos uusi tieto tallennetaan samalla avaimella. Koodeista voidaan huomata, että tiedon tallennus voidaan toteuttaa varsin yksinkertaisesti ja molempien tietokantojen kohdalla koodit ovat hyvin samankaltaisia.

Lopullisesti tietokannan valinnassa määrävä tekijä oli se, että toimeksiantaja käytti Riakia eräässä toisessa projektissa ja sitä varten oli tarve kerätä osaamista ja kokemusta Riakin käytöstä Node.js-ympäristössä. Couchbasella olisi kyllä ollut tarjottavana parempi hallintatyökalu ja selkeämpi Map/Reduce-toimintojen toteutus, mutta kyseiset ominaisuudet eivät olleet merkitseviä, koska varsinkin Couchbase-lisäkirjaston toimivuus Node.js:ssä ei ollut vaaditulla tasolla. Couchbasen dokumentaatio oli selkeästi kattavampi kuin Riakissa, koska siinä kuvataan teknisten asioiden lisäksi myös oikea tapa toimia. Riakin dokumentaatio kuvaa asiat tekniseltä kannalta ja oli paikoin varsin hankalasti ymmärrettävä.

5.3 Noutotoiminto

Alkuperäinen noutotoiminto oli toteutettu Pythonilla ja se oli rakenteeltaan yksinkertainen eli siihen ei ollut toteutettu aineiston pilkkomista kentiksi eikä tallennusta tietokantaan. Noutotoimintoon oli ainoastaan toteutettu hankintailmoitusten tallennus XML-tiedostoiksi. Puuttuvien toimintojen lisäksi sovellus päätettiin toteuttaa Node.js:llä.

Toiminnon vaatimuksena oli hankintailmoitusten lataus siten, että voidaan konfiguroida, kuinka monta päivää taaksepäin ladataan ilmoituksia. Ilmoitukset tulee tallentaa yksittäisinä XML-dokumentteina omaan hakemistoon siten, että hakemiston nimi on muotoa vuosi/kk ja tiedostonnimi on ilmoitusnumero. Ilmoituksesta tulee poimia tietokantatallennusta varten ainakin päivämäärä, ilmoitusnumero, ilmoituslaji, julkaisija, otsikko, CPV-koodi, arvo ja määräaika. Liitetiedostot tulee ladata hakemistoon /vuosi/kk/ilmoitusnumero/liitenimi. Vaatimusten mukaisesti toiminnon toteutus jaettiin pienempiin alitehtäviin:

1. XML-raaka-aineiston lataus
2. Raaka-aineiston jako ilmoituksiksi
3. JSON-objektin muodostaminen XML-muotoisista ilmoituksista

4. Tallennus Riakiin
5. Liitetiedostojen lataus ja tallennus

Noutotoiminnon suunnittelu aloitettiin perehtymällä Editan toimittamiin teknisiin aineistoihin virallisen hankintailmoituspalvelun latauspalvelusta. Noudettavien XML-dokumenttien rakenne oli hyvin monimutkainen, jolloin niiden rakennetta kuvaava XSD-skeema oli myös monimutkainen. XML-rakenne selvisi parhaiten tutkimalla rakennetta suoraan XML-tiedostosta ja käyttämällä XSD-skeemaa tukena.

XML-muotoinen raaka-aineisto toteutettiin lataamalla se Node.js:n request-lisäkirjastolla verkko-osoitteesta seuraavan mallin mukaisesti:

```
http://www.hankintailmoitukset.fi/fi/notice/export?
get&start=YYYYMMDDhhmmss&end=YYYYMMDDhhmmss
```

Osoitteelle annettiin parametreinä alkupäivämäärä ja kellonaika sekä loppupäivämäärä ja kellonaika. Pyyntö palauttaa XML-dokumentin, joka sisältää kaikki ilmoitukset, jotka ovat ilmestyneet parametrinä annetulla aikavälillä, Yksi ilmoitus on aina XML-elementin <WRAPPED_NOTICE> sisällä. Liitteessä 4 on mallina yhden hankintailmoituksen XML-rakenne. Jokainen ilmoitus poimittiin taulukkoon koodilla:

```
var notice_nodes =
xpath.select('/NOTICES/WRAPPED_NOTICE',doc);
```

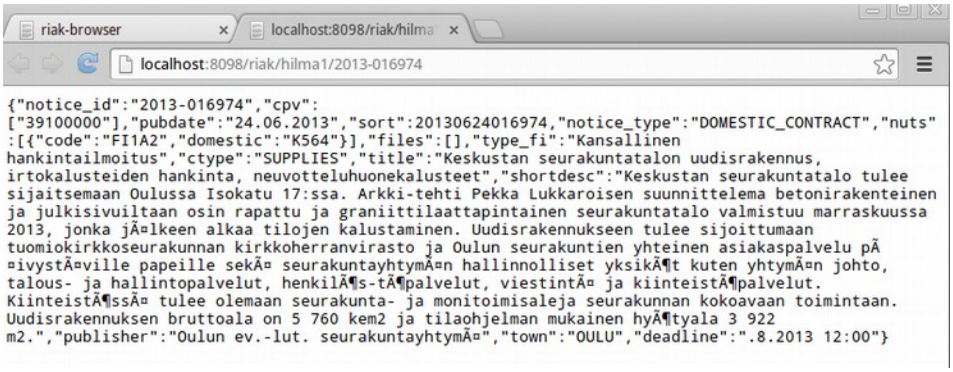
Jokaiselle ilmoitukselle täytyi suorittaa tietojen poiminta XML-tiedostosta JSON-objektin kentiksi. Sovellukseen tarvittiin siten lisäkirjastoja, joilla voitaisiin käsitellä XML-tiedostoja. Muutamien kokeilujen jälkeen päädyttiin xmldom- ja xpath-lisäkirjastoihin. Hankintailmoituksen pääelementin nimi oli riippuvainen hankintailmoituksen tyypistä taulukon 1. englanninkielisen nimityksen mukaan. Tämän lisäksi jokaisen tyypin XML-rakenne oli yksilöllinen. Tällä tavoin määritelty rakenne vaati sen, että poimittavia kenttiä varten tuli määritellä JSON-muotoinen konfiguraatiotiedosto xml-mapping.json, jossa oli määritelty jokaisen tarvittavan hankintailmoitustyypin kentille hakupolku Xpath-muodossa. Poikkeuksena oli CPV-lisäkoodit ja NUTS-aluekoodit, joiden hakupolku määritettiin suoraan ohjelmakoodiin. Hakupolkuja määriteltäessä huomattiin, että hankinnan arvoa ei ollut mahdollista poimia yksilöllisesti, joten ominaisuutta ei toteutettu vielä tässä vaiheessa. Liitteessä 5 on esitelty yhden hankintailmoitustyypin määritys xml-mapping.json-tiedostossa.

Hankintailmoituksen JSON-objektin tallennus Riakiin tehtiin liitteessä 3 esitetyllä koodilla, jossa hankintailmoitus JSON-objekti on parametrinä. Riak-tietueen avaimena on hankintailmoituksen numero, joka on yksilölli-

nen jokaiselle hankintailmoitukselle. Yksittäisen ilmoituksen XML-rakenteen tallennus tiedostoon tehtiin siten, että se tallennetaan sellaisenaan määrittelyn mukaisesti. Liitetiedostojen osoitteet poimittiin JSON-objektin luonnin yhteydessä taulukkona ja tehtiin erillinen toiminto, joka lataa liitteet sellaisenaan määrittelyn mukaisesti.

Alustavassa versiossa noutotoiminto toimi oikein, kun se suoritettiin hakemaan kuluvan päivän ilmoitukset. Tietojen haku useammalle päivälle aiheutti timeout-virheitä ja lopullinen syy ongelmaan oli sovelluksen liian nopea toiminta. Sekä raaka-aineiston haku että liitetiedostojen lataus käyttävät request-lisäkirjastoa ja tällöin kyseessä on asynkronisten toimintojen suoritus, jolloin latauksia tulee samaan aikaan erittäin paljon. Tästä syystä ennen kuin yhden päivän tieto on ladattu, ohjelma suorittaa jo seuraavaa latausoperaatiota. Vastaava ongelma oli myös liitetiedostojen tallennuksessa, jolloin sovellus kaatui, koska se oli avannut liikaa tiedostoja. Ongelmat korjattiin määrittelemällä yhden ilmoituksen käsittelytoiminnot omaan funktioon, joka kutsuu itseään rekursiivisesti, kun edellinen operaatio on suoritettu loppuun. Rekursiivinen funktio ottaa käsiteltäväksi aina seuraavan ilmoituksen, kunnes koko taulukko on käyty läpi. Ongelman ratkaisuun testattiin myös async-lisäkirjaston käyttöä, joka mahdollistaisi peräkkäisen suorituksen, mutta se ei ratkaisut ongelmaa.

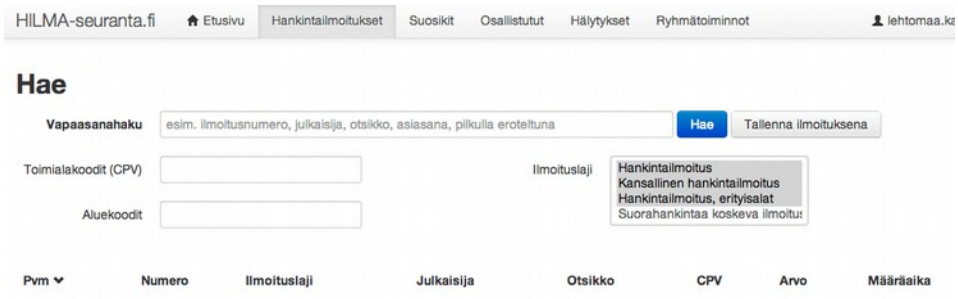
Noutotoimintoon tehtiin pieniä muutoksia myös haku- ja ilmoitustoiminnon toteutuksen yhteydessä ja samalla noutotoiminnon ohjelmakoodi siirrettiin itse Hilma-sovelluksen alle omaan hakemistoon. Kuvassa 14 on esitetty yhden noutotoiminnolla tallennetun hankintailmoituksen tietojen rakenne Riakissa.



Kuva 14. Tallennettu hankintailmoitus Riakissa.

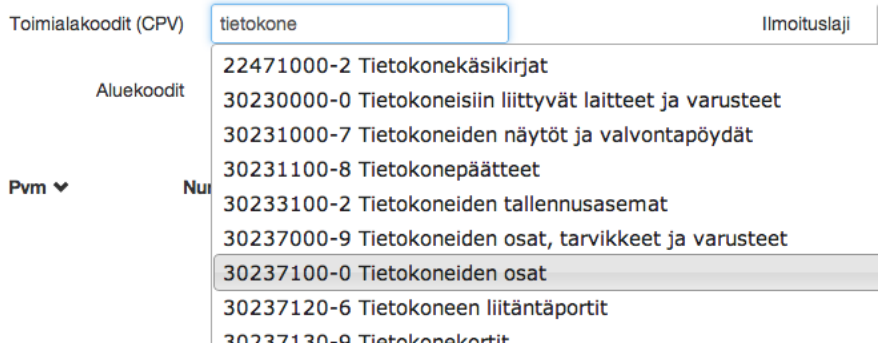
5.4 Hakutoiminto

Hakutoimintoa varten uuteen hankintailmoitusportaaliin oli jo toteutettu kuvassa 15 oleva hakusivu, mutta siitä puuttui vielä itse toiminnallisuus. Hakutoiminto toteutettiin siten, että hakupyyntö lähetetään AJAX-pyyntönä, jolla on parametrinä syötetyt hakutekijät JSON-muotoisena objektina. Pyyntö palauttaa JSON-objektin, josta generoidaan sisältö Javaskriptillä tulostaulukkoon.



Kuva 15. Portaalin hakusivu.

Lomakkeen hakukenttiin toimialakoodit eli CPV ja aluekoodi eli NUTS, toteutettiin automaattitäydennys Jquery UI:n autocomplete-toiminnolla. Kun CPV-kenttään alkaa syöttää koodia tai sitä kuvaavaa tekstiä, niin kenttään haetaan koodi ja teksti, kuten kuvassa 16 esitetään. Vastaavasti toteutettiin myös aluekoodikenttä. Automaattitäydennyskentissä tuli olla mahdollista valita useampi kuin yksi koodi.



Kuva 16. Automaattitäydennys toimialakoodi-kentässä.

Itse hakulogiikka toteutettiin palvelimelle ja hyvin aikaisessa vaiheessa huomattiin, että Riakin hakumoottoria ei voida käyttää vaan haku oli tehtävä Map/Reducella. Tämä johtui siitä, että NUTS- ja CPV-koodit tallennetaan Riakissa taulukkona hankintailmoitustietueen sisään, kuten kuvasta 14 voidaan havaita. Sekä CPV- ja NUTS-koodit ovat hierarkkisia esimerkin 11 mukaan, jossa ylemmän tason code on aina alemman tason domestic. Vaatimuksena oli, että kun käyttäjä syöttää ylemmän tason koodin, niin itse hakua varten generoidaan kaikki sitä alemman tason koodit mu-

kaan hakuun. Sekä automaattitäydennystä että hakua varten luotiin käsin JSON-tiedostot CPV:lle ja NUTS:lle.

```
{"code":"FI1","domestic":"","text":"Manner-Suomi"},
{"code":"FI1A","domestic":"FI1","text":"Pohjois-Suomi"},
{"code":"FI1A3","domestic":"FI1A","text":"Lappi"},
{"code":"S191","domestic":"FI1A3","text":"Rovaniemen
seutu"},
{"code":"K698","domestic":"S191","text":"Rovaniemi"}
```

Esimerkki 11. NUTS hierarkkinen rakenne.

Hakukoodista tuli varsin monimutkainen, kuten liitteestä 6 voidaan havaita. Koodissa map-funtio saa arg-parametrina taulukon käyttäjän syöttämistä hakutekijöistä. Funktion alussa yhdistetään tekstikentät ja etsitään osumaa vapaasanahaku-kentän sanoista. Sen jälkeen tarkistetaan CPV- ja NUTS-koodit ja lopuksi tarkistetaan hankintailmoitustyyppi. Jos kyseinen tietue kohdistuu hakutekijöihin niin se palauttaa tietueen datan Reduce-vaiheeseen

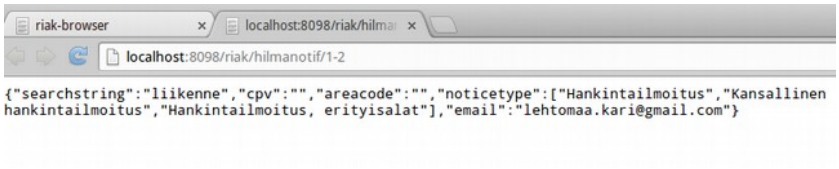
Hakutoiminto etsii vapaasanahaku-kentän tekstiä kaikista tallennetuista tekstikentistä. Noutotoimintoon lisättiin myös lyhyen kuvauskentän tallennus Riakiin. Tietojen järjestämistä varten lisättiin sort-kenttä, joka koostuu päivämäärästä ja hankintailmoitusnumerosta. Riakin Map/Reducessa ei ole oletuksena tulosjoukon järjestämistä, joten se lisättiin Reduce-vaiheeseen. Tulosjoukon järjestämiskoodi kopioitiin Riak-contrib-sivustolta ja siihen tehtiin vain pieniä tarvittavia muutoksia.

5.5 Ilmoitustoiminto

Toiminnon vaatimuksena oli, että käyttäjä voi määritellä haun tallennettavaksi ilmoitukseksi. Käyttäjän kirjauduttua hankintailmoitusportaalin sivulla näytetään lista, jossa näkyvät ne hankintailmoitukset, jotka ovat ilmestyneet ilmoituksen luonnin jälkeen. Käyttäjälle lähetetään myös sähköposti uusista hankintailmoituksista ilmoitustoiminnon mukaisesti.

Ilmoitustoiminto toteutettiin siten, että se toimii kuten hakutoiminto ja käyttää paljolti samaa koodia. Erona on se, että hankintailmoitusten osuvuus tarkistetaan noutotoiminnossa. Käyttäjän tallettaessa haun ilmoituksena luodaan Riakiin omaan bucketiinsa tietue kuvan 17. mukaisesti, jossa tietueen avaimena on käyttäjän id, alaviiva ja juokseva numero.

Noutotoiminto tarkistaa jokaisen noudetun hankintailmoituksen yhteydessä sen osuvuutta Riakista löytyviin ilmoituksiin. Jos hankintailmoitus osuu kohdalle, niin Riakiin tallennetaan omaan bucketiin osuma kuvan 18 mukaisesti. Samalla käyttäjälle lähetetään sähköposti. Käyttäjän kirjautuessa portaaliin haetaan osumien mukaisten hankintailmoitusten tiedot aloitussivulle.



Kuva 17. Tallennettu ilmoitus Riakissa.

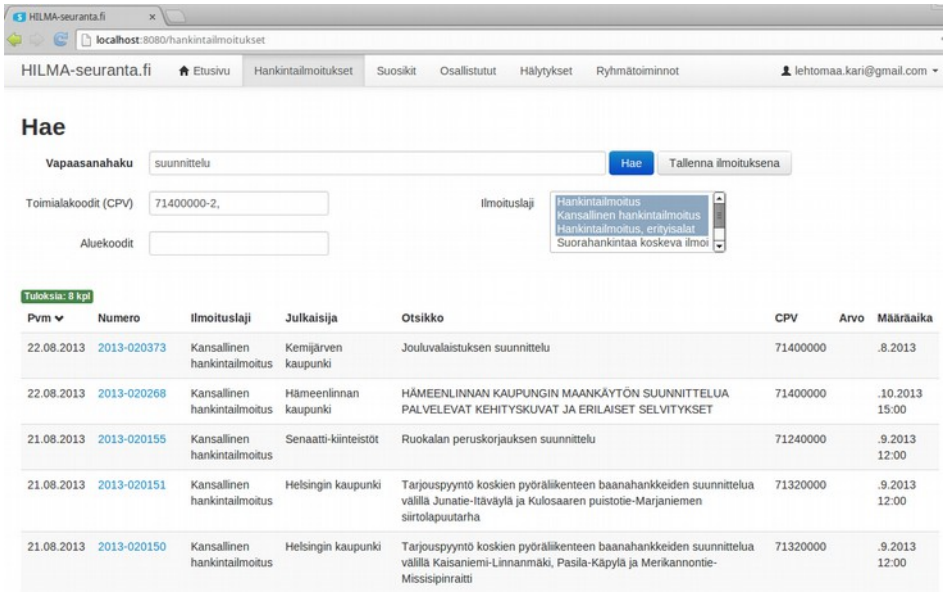
Kuva 18. Tallennettu ilmoituksen osuma Riakissa.

5.6 Hajautuksen testaus

Sovelluksen toimintaa hajautetulla tietokannalla testattiin luomalla kahden Riak-palvelimen klusteri. Testikoneina oli kone 1 ja kone 2, joissa koneen 1 IP-osoite oli 192.168.1.107 ja koneen 2 IP-osoite oli 192.168.1.7. Koneella 1 oli myös asennettuna uusi noutosovellus ja hankintailmoitusportaali-palvelin. Molempien koneiden Riak konfiguraatioon määritettiin ulkoisen ethernet-liitännän osoite ja tarkistettiin, että muut asetukset samalla tavalla molemmissa konfiguraatioissa. Kun molemmilla koneilla Riak oli käynnissä uudella konfiguraatiolla, niin kone 2 liitettiin koneen 1 ympäristöön komennoilla:

```
bin/riak-admin cluster join riak@192.168.1.107
bin/riak-admin cluster plan
bin/riak-admin cluster commit
```

Kahdelle palvelimelle hajautettu ympäristö testattiin käynnistämällä ensin noutotoiminto, joka toimi ongelmitta. Lopuksi käynnistettiin portaali ja testattiin hakua, joka toimi silmin nähden nopeammin kuin pelkästään yhdellä tietokantapalvelimella. Nopeutumisen syynä oli se, että aikaa vievä Map/Reduce suoritettiin nyt kahdella koneella. Kuvassa 19 on näkymä portaalissa haun jälkeen.



Kuva 19. Hakunäkymä hankintailmoitusportaalissa hajautetulla tietokannalla.

5.7 Yhteenveto toteutuksesta

Toimintojen toteutus oli varsin haastavaa ja varsinkin noutotoiminnon yhteydessä tuli esiin ongelma asynkronisen koodin kanssa ja sen selvittelyyn meni varsin paljon aikaa. Hakutoiminnosta tuli varsin monimutkaisen näköinen ja sen yhteydessä tuli esiin Node.js koodin huono luettavuus monimutkaisten toimintojen yhteydessä. Selvyyden vuoksi koodia jaettiin omiin funktioihin ja lisäkirjastoihin, mutta se ei kokonaan selkeyttänyt koodia. Esimerkiksi Map/Reduce koodi tulee olla yhdessä palassa, koska se suoritetaan Riakissa eikä Node.js:ssä.

Riakin ja Node.js:n yhteydessä tuli myös ilmi dokumentaation puutteellisuus. Varsinkin lisäkirjastojen dokumentaatio oli varsin puutteellista ja monet asiat tuli selvittää lukemalla lisäkirjastojen lähdekoodia ja yksinkertaisesti kokeilemalla. Syy puutteelliseen dokumentaatioon on tuotteiden keskeneräisyys ja se että ne kehittyvät nopealla tahdilla, jolloin dokumentaatiota ei välttämättä pidetä yllä samaa tahtia. Suurena etuna Node.js:n käytössä on se että samaa kieltä käytetään niin palvelin- kuin selainpuolen toimintojen toteutuksessa.

Vaikka vaaditut toiminnot toteutettiin, niin ne eivät ole vielä tuotantokäyttö kelpoisia. Käytännön testeissä haku- ja ilmoitustoiminto toimivat vaatimusten mukaisesti, mutta monimutkaisilla hakutermeillä haku oli varsin hidas. Haun hitaus on varsin ymmärrettävää johtuen CPV- ja NUTS-koodien rakenteesta. Haun optimointia ei vielä suoritettu, koska kehityksen aikaiset testit suoritettiin yhdellä Riak-solmukoneella, jolloin suorituskyky ei vastaa tuotantoympäristöä.

6 YHTEENVETO

Työssä selvitettiin NoSQL-tietokantojen keskeiset periaatteet ja niiden perusteella saatiin jonkinlainen käsitys siitä, että mihin tämäntyyppiset tietokannat sopivat. NoSQL-tietokantoihin liittyy paljon syvällistä teoriaa ja lisäksi eri valmistajien tuotteissa on toiminnallisia eroja, joihin tässä työssä ei ollut mahdollista syvemmin paneutua. NoSQL-tietokannat elävät ihan omassa maailmassaan ja suurinpiirtein kaikki mitä liittyy relaatiotietokantoihin voidaan unohtaa. NoSQL-tietokannat antavat sovelluskehittäjälle enemmän vapauksia, koska tietorakennetta voi muuttaa kehityksen aikana.

Node.js todettiin sopivaksi verkkosovellusten kehitykseen, vaikka sovellusten toteuttaminen ei ole ongelmatonta. Noutotoiminnon kohdalla kuitenkin huomattiin, että Node.js ei sovellu ihan jokaiseen tilanteeseen. Node.js on nopeasti kehittyvä ja se näkyy dokumentaation puutteena varsinkin lisäkirjastojen osalta. Node.js:n asynkroninen toiminta tekee koodin toiminnasta tehokasta, mutta koodin luettavuus kärsii varsinkin isompien projektien osalta. Hyvänä puolena verkkosovellusten kehityksen kannalta on yhden kielen käyttö palvelimessa ja selaimessa. Node.js:ssä toteutetut sovellukset ovat myös nopeita ja kuluttavat vähän muistia.

Käytännön osuudessa valittiin sopiva tietokanta uudelle hankintailmoitusportaalille. Tietokannaksi valittiin Riak käytännössä sen vuoksi, että se oli paremmin tuettu Node.js:llä ja toimeksiantaja oli ottanut sen käyttöön aiemmin toisessa projektissa. Noutotoiminto oli varsin työläs toteuttaa johtuen lähinnä Node.js:n sopimattomuuudesta tämän tyyliseen skriptiohjelmointiin. Node.js:n asynkroninen tapa toimia ei sopinut ilman virityksiä sarjatoimintoihin. Haku- ja ilmoitustoiminnon toteutus oli helpompaa, mutta niissä työläin ja haasteellisin asia oli hakujen toteuttaminen. Tuloksena oli varsin monimutkainen Map/Reduce-koodi.

Työssä vaaditut asiat saatiin selvitettyä sekä toiminnot saatiin toteutettua. Työssä testattiin myös hajautuksen toteutus, jonka tekeminen todettiin erittäin yksinkertaiseksi. Toteutetut toiminnot eivät kuitenkaan vielä ole valmiita tuotantoon, koska ne vaativat vielä tehostamista. Esimerkiksi hankintailmoituksen avaamista ei toteutettu tässä vaiheessa, koska toteutus olisi ollut suuritöinen johtuen XML-tiedostojen erilaisesta rakenteesta. Haku- ja ilmoitustoiminnon yhteyteen harkittiin sanojen samankaltaisuuden vertailua, mutta se jätettiin pois tästä vaiheesta. Myös liitetiedostojen indeksointia hakua varten ei vielä toteutettu. Hankintailmoitukset on varsin hyvin kuvailtu esimerkiksi lyhytkuvaus-kentässä, jolloin haun kattavuus on varsin hyvä vielä tässä vaiheessa. Toimeksiantaja jatkaa palvelinkehitystä ja hajautetun tietokannan ansiosta palvelu voidaan laajentaa koskemaan myös EU:n hankintailmoituksia.

7 LÄHTEET

Adamo D. 2011. NoSQL databases advantages and disadvantages. 28.06.2011. Viitattu 17.05.2013. http://tekedia.com/12083/nosql-database-advantages-and-disadvantages/

Anderson J. C, Lehnardt J & Slater N. n.d. CouchDB definitive guide draft, documents. Viitattu 19.05.2013. http://guide.couchdb.org/draft/documents.html

Capan T. 2013. Why the hell would I use Node.js. Viitattu 28.08.2013. http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

Couchbase. 2013. Couchbase server manual. Viitattu 09.07.2013. http://www.couchbase.com/docs/couchbase-manual-2.1.0/index.html

Fowler M. 2013. Luentovideo, NoSQL distilled to an hour. Viitattu 23.05.2013.http://vimeo.com/66052102?utm_source=NoSQL+Weekly+Newsletter&utm_campaign=5725e1eb7d-NoSQL_Weekly_Issue_130_May_23_2013&utm_medium=email&utm_term=0_2f0470315b-5725e1eb7d-199631801

Gupta V. 2010. NoSQL databases , part 1 – landscape. 05.01.2010. Viitattu 12.5.2013. http://www.vineetgupta.com/2010/01/nosql-databases-part-1-landscape/

Harrison G. 2013. Consistency_Models_in_Non-Relational_Databases. Viitattu 23.05.2013. http://dbpedias.com/wiki/NoSQL:Consistency_Models_in_Non-Relational_Databases

Hankintalaki. Laki julkisista hankinnoista 6:35§. 30.3.2007/348.

HBase. n.d. 2012. Hbase datamodel. Viitattu 19.05.2013. http://hbase.apache.org/book.html#datamodel

HILMA. n.d. Hankintailmoitusportaali. Viitattu 27.04.2013. http://www.hankintailmoitukset.fi/fi/.

JSON. n.d. Introducing JSON. Viitattu 02.08.2013. http://www.json.org/

Julkinen hankinta. n.d. Yleistä julkisista hankinnoista. Viitattu 27.04.2013. http://www.hankintailmoitukset.fi/fi/docs/yleista/.

Katsov I. 2012. NoSQL data modelling techniques. Viitattu 30.08.2013. http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-

techniques/

Neo4J. n.d. What is a Graph database? Viitattu 30.08.2013. http://www.neo4j.org/learn/graphdatabase.

Node.js. 2013. Node.js. Viitattu 02.07.2013. http://nodejs.org/.

NoSQL. n.d. NoSQL Databases. Viitattu 12.05.2013. http://nosql-database.org/

Nurse C. 2013. Map reduce explained. Viitattu 19.06.2013. http://www.charlesnurse.com/Blog/tabid/226/EntryId/35/Look-Mom-NoSQL-7-Map-Reduce-explained.aspx

Rajapintamäärittely. Lisäarvopalvelun tarjoajien rajapinnan kuvaus, EDITA 2011.

Riak. 2013. Basho Riak docs. Viitattu 09.08.2013. http://docs.basho.com/riak/latest/.

Spencer N. 2012. How much data is created every minute? 19.06.2012. Viitattu 12.05.2013. http://www.visualnews.com/2012/06/19/how-much-data-created-every-minute/.

Strauch C. NoSQL Databases, pdf. Viitattu 30.06.2013. http://www.christof-strauch.de/nosqldbs.pdf

Teixeira P. 2013. Professional Node.js. Indianapolis USA, John Wiley & Sons Inc.

Valtioneuvosto, Valtioneuvoston asetus julkisista hankinnoista nro. 614/2007. 24.5.2007.

Vogel L. 2013. Git tutorial version 5.3. Viitattu 19.08.2013. http://www.vogella.com/articles/Git/article.html.

Vogels W. 2007. All things distributed, eventually consistent. Viitattu 23.0.5.2013. http://www.allthingsdistributed.com/2007/12/eventually_consistent.html

Why NoSQL. Couchbase, why NoSQL. Viitattu 14.05.2013. http://www.couchbase.com/why-nosql/nosql-database

NODE.JS EXPRESS-MALLI

```
var express = require('express');
var app = express();


app.set('view engine', 'html')
app.set('partials', {header: "header"});
app.enable('view cache');
app.engine ('html', require('hogan-express'));
app.set('views', __dirname + '/views');


console.log("Running  server");

app.use(express.errorHandler({ dumpExceptions: true, showS-
tack: true }));


app.use(express.cookieParser());
app.use(express.cookieSession({ secret: "qwerty1234"}));


app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(express.static(__dirname + '/public'));


// ignore logging of static files
app.use(express.logger());


app.get('/', function(req, res){
  res.locals = {data1: "data1", data2: "data2"};
  res.render ("index", {partials: {body: 'body'}});

});


app.listen(8080);
console.log('Listening on port 8080');
```

TIEDONTALLENNUS COUCHBASEEN NODE.JS:LLÄ.

```
// load the Couchbase driver and connect to the cluster
var cb = require('couchbase');

var settings = {
    "debug" : false,
    "user" : "Administrator",
    "password" : "xxxxx",
    "hosts" : [ "192.168.1.3:8091" ],
    "bucket" : "hilma1"
};

exports.saveToCB=function(doc){
    cb.connect(settings, function(err, bucket) {
    if (err) {
           throw(err); }


           bucket.add(doc.notice_id, doc, function(err,meta)
{
                       if(err){
                           console.log('Error occured in
Couchbase save');
                       }
                       else {
                           console.log("data inserted");
                           console.log(JSON.stringify(meta));
                       }
           });
           });
}
```

TIEDONTALLENNUS RIAKIIN NODE.JS:LLÄ.

```
var config = require('../config.json');

var riakdb = require('riak-js').getClient({host: con-
fig.riakhost, port: config.riakport});

exports.save = function(key, data){
           bucket = config.riakbucket;
           riakdb.save(bucket,key,data,
function(err,buff,meta){
                       if(err){
                           console.log("Fatal error, Riak not
available, cannot continue");
                           process.exit(-1);
                       }
                       else { console.log("Saving to Riak " +
key);}

           });
           }
```

HANKINTAILMOITUS XML-MUODOSSA.

```
<WRAPPED_NOTICE>
   <ID>2013-008205</ID>
   <MODIFIED>
     <DAY>22</DAY>
     <MONTH>03</MONTH>
     <YEAR>2013</YEAR>
     <TIME>14:18:59</TIME>
   </MODIFIED>
   <CONTRACT LG="FI" CATEGORY="ORIGINAL" FORM="2" VERSION="R2.0.8.S02">
     <FD_CONTRACT CTYPE="SUPPLIES">
       <CONTRACTING_AUTHORITY_INFORMATION>
         <NAME_ADDRESSES_CONTACT_CONTRACT>
           <CA_CE_CONCESSIONAIRE_PROFILE>
             <ORGANISATION>
               <OFFICIALNAME>IS-Hankinta Oy</OFFICIALNAME>
             </ORGANISATION>
             <ADDRESS>Viestikatu 3, 3.krs</ADDRESS>
             <TOWN>Kuopio</TOWN>
             <POSTAL_CODE>70600</POSTAL_CODE>
             <COUNTRY VALUE="FI"/>
             <PHONE>+358 447182912</PHONE>
           </CA_CE_CONCESSIONAIRE_PROFILE>
           <TENDERS_REQUESTS_APPLICATIONS_MUST_BE_SENT_TO>
             <CONTACT_DATA>
               <ORGANISATION>
                 <OFFICIALNAME>IS-Hankinta Oy</OFFICIALNAME>
               </ORGANISATION>
               <URL>https://tarjouspalvelu.fi/ishankinta/?id=8201...</URL>
             </CONTACT_DATA>
           </TENDERS_REQUESTS_APPLICATIONS_MUST_BE_SENT_TO>
         </NAME_ADDRESSES_CONTACT_CONTRACT>
       </CONTRACTING_AUTHORITY_INFORMATION>
       <OBJECT_CONTRACT_INFORMATION>
         <DESCRIPTION_CONTRACT_INFORMATION>
           <TITLE_CONTRACT>Lyövän sydämen ohituskirurgiassa k...</TITLE_CONTRACT>
           <TYPE_CONTRACT_LOCATION>
             <TYPE_CONTRACT VALUE="SUPPLIES"/>
```

```
    </TYPE_CONTRACT_LOCATION>
    <SHORT_CONTRACT_DESCRIPTION>
      <P>IS-Hankinta ....</P>
      <P>https://tarjouspalvelu.fi/ishankinta/?id=8201...</P>
    </SHORT_CONTRACT_DESCRIPTION>
    <CPV>
      <CPV_MAIN>
        <CPV_CODE CODE="33160000"/>
      </CPV_MAIN>
    </CPV>
    <F02_DIVISION_INTO_LOTS>
      <F02_DIV_INTO_LOT_YES VALUE="ALL_LOTS"/>
    </F02_DIVISION_INTO_LOTS>
    <ACCEPTED_VARIANTS VALUE="NO"/>
  </DESCRIPTION_CONTRACT_INFORMATION>
  <QUANTITY_SCOPE>
    <NO_OPTIONS/>
  </QUANTITY_SCOPE>
</OBJECT_CONTRACT_INFORMATION>
<LEFTI_CONTRACT/>
<PROCEDURE_DEFINITION_CONTRACT_NOTICE>
  <TYPE_OF_PROCEDURE>
    <TYPE_OF_PROCEDURE_DETAIL_FOR_CONTRACT_NOTICE>
      <PT_OPEN/>
    </TYPE_OF_PROCEDURE_DETAIL_FOR_CONTRACT_NOTICE>
  </TYPE_OF_PROCEDURE>
  <ADMINISTRATIVE_INFORMATION_CONTRACT_NOTICE>
    <RECEIPT_LIMIT_DATE>
      <DAY>6</DAY>
      <MONTH>5</MONTH>
      <YEAR>2013</YEAR>
      <TIME>14:00</TIME>
    </RECEIPT_LIMIT_DATE>
    <LANGUAGE>
      <LANGUAGE_EC VALUE="FI"/>
    </LANGUAGE>
  </ADMINISTRATIVE_INFORMATION_CONTRACT_NOTICE>
</PROCEDURE_DEFINITION_CONTRACT_NOTICE>
<COMPLEMENTARY_INFORMATION_CONTRACT_NOTICE>
  <NOTICE_DISPATCH_DATE>
```

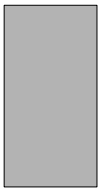
```
            <DAY>22</DAY>
            <MONTH>3</MONTH>
            <YEAR>2013</YEAR>
          </NOTICE_DISPATCH_DATE>
        </COMPLEMENTARY_INFORMATION_CONTRACT_NOTICE>
        <ATTACHMENTS>
          <ATTACHMENT TYPE="EXTERNAL">https://tarjouspalvelu.fi/ishankinta/?id=8201...</ATTACHMENT>
        </ATTACHMENTS>
      </FD_CONTRACT>
    </CONTRACT>
</WRAPPED_NOTICE>
```

MÄÄRITYS XML-MAPPING.JSON TIEDOSTOSSA.

```
{
  "CONTRACT":{
    "typefi": "Hankintailmoitus",
    "ctype": "/FD_CONTRACT/@CTYPE",
    "title": "/FD_CONTRACT/OBJECT_CONTRACT_INFORMATION/DESCRIPTION_CONTRACT_INFORMATION/TITLE_CONTRACT",
    "shortdesc":
"/FD_CONTRACT/OBJECT_CONTRACT_INFORMATION/DESCRIPTION_CONTRACT_INFORMATION/SHORT_CONTRACT_DESCRIPTION/*",
    "cpv":
"/FD_CONTRACT/OBJECT_CONTRACT_INFORMATION/DESCRIPTION_CONTRACT_INFORMATION/CPV/CPV_MAIN/CPV_CODE/@CODE",
    "publisher": "/FD_CONTRACT/CONTRACTING_AUTHORITY_INFORMATION/NAME_ADDRESSES_CONTACT_CONTRACT/CA_CE_CONCES-
SIONAIRE_PROFILE/ORGANISATION/OFFICIALNAME",
    "town": "/FD_CONTRACT/CONTRACTING_AUTHORITY_INFORMATION/NAME_ADDRESSES_CONTACT_CONTRACT/CA_CE_CONCESSIO-
NAIRE_PROFILE/TOWN",
    "deadlineday": "/FD_CONTRACT/PROCEDURE_DEFINITION_CONTRACT_NOTICE/ADMINISTRATIVE_INFORMATION_CONTRACT_NO-
TICE/RECEIPT_LIMIT_DATE/DAY",
    "deadlinemonth":
"/FD_CONTRACT/PROCEDURE_DEFINITION_CONTRACT_NOTICE/ADMINISTRATIVE_INFORMATION_CONTRACT_NOTICE/RECEIPT_LIMIT_DA
TE/MONTH",
    "deadlineyear": "/FD_CONTRACT/PROCEDURE_DEFINITION_CONTRACT_NOTICE/ADMINISTRATIVE_INFORMATION_CONTRACT_NO-
TICE/RECEIPT_LIMIT_DATE/YEAR",
    "deadlinetime": "/FD_CONTRACT/PROCEDURE_DEFINITION_CONTRACT_NOTICE/ADMINISTRATIVE_INFORMATION_CONTRACT_NO-
TICE/RECEIPT_LIMIT_DATE/TIME",
    "attachment": "/FD_CONTRACT/ATTACHMENTS"
  }, ...
```

HAKUTOIMINNON MAP/REDUCE

```
db.mapreduce.add(BUCKET)
  .map(function(value, keyData, arg) {
        var data = Riak.mapValuesJson(value)[0];

        // build text search str
        var wsfields = (value.key + " " + data.title + " "
+ data.shortdesc + " " + data.publisher).toLowerCase();


        // test for matching word
        var wordmatch = false;


        if(arg.words.length == 0) wordmatch = true;
        else {
          while(arg.words.length > 0){
            if(wsfields.indexOf( arg.words.pop() ) > -1 ){
              wordmatch = true;
              break;
              } // if
            } // while
          } // else


        // test for matching cpv
       var cpvmatch = false;
       if(arg.cpv.length == 0) cpvmatch = true; // no cpvs
given
       else {
          while(arg.cpv.length > 0){ // Find matching and
remove it from param (arg) array
            if(data.cpv.indexOf(arg.cpv.pop() ) > -1) {
              cpvmatch =true;
              break;
            } // if
          } // while
        } // else


         // test for matching nuts
       var nutsmatch = false;
       if(arg.nuts.length == 0) nutsmatch = true; // no
nuts given
       else {
          while(arg.nuts.length > 0){ // Find matching and
remove it from param (arg) array
            var curnuts = arg.nuts.pop();


            for(var d = 0; d < data.nuts.length; d++){
```



```
              if(data.nuts[d].code == curnuts ||
data.nuts[d].domestic == curnuts){
                nutsmatch =true;
                break;
                } // if
              }
            } // while
          } // else


          // test for matching noticetype (in finnish)
          var ntmatch = false;
          if(arg.noticetype.length == 0) ntmatch = true; //
no type given
          else if(arg.noticetype.indexOf(data.type_fi) > -1)
ntmatch = true;


         // search matching
          if(wordmatch && cpvmatch && ntmatch && nutsmatch){
             return [data];
             }
          else { return []; }

}, searchobject) // end of map


  // sort function from Riak contrib

  .reduce(function(values, arg) {
       var field = (typeof arg === "undefined" || arg ===
null) ? undefined : arg.by;
       var reverse = ((typeof arg === "undefined" || arg
=== null) ?
      undefined : arg.order) === 'desc';
       values.sort(function(a, b) {
         if (reverse) {
           var _ref = [b, a];
           a = _ref[0];
           b = _ref[1];
         }
         if (((typeof a === "undefined" || a === null) ?
Undefined :
      a[field]) < ((typeof b === "undefined" || b ===
null) ? Undefined :
      b[field])) {
           return -1;
         } else if (((typeof a === "undefined" || a ===
null) ? Undefined :
      a[field]) === ((typeof b === "undefined" || b ===
null) ? Undefined :
      b[field])) {
           return 0;
         } else if (((typeof a === "undefined" || a ===
null) ? Undefined :
      a[field]) > ((typeof b === "undefined" || b ===
null) ? Undefined :
      b[field])) {
           return 1;
```

```
        }
      });
      return values;
    }, { by: 'sort', order: 'desc' })
.run(
        function(err, noticedata){
    callback(noticedata);
        } );
```